Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2010, Article ID 243985, 16 pages
doi:10.1155/2010/243985

Research Article

Efficient Certification Path Discovery for MANET

Georgios Kambourakis, Elisavet Konstantinou, Anastasia Douma, Marios Anagnostopoulos, and Georgios Fotiadis

Info-Sec-Lab Laboratory of Information and Communications Systems Security, University of the Aegean, 83200 Samos, Greece

Correspondence should be addressed to Georgios Kambourakis, gkamb@aegean.gr

Received 1 February 2010; Accepted 8 April 2010

Academic Editor: Nicholas Kolokotronis

A Mobile Ad Hoc Network (MANET) is characterized by the lack of any infrastructure, absence of any kind of centralized administration, frequent mobility of nodes, network partitioning, and wireless connections. These properties make traditional wireline security solutions not straightforwardly applicable in MANETs, and of course, constitute the establishment of a Public Key Infrastructure (PKI) in such networks a cumbersome task. After surveying related work, we propose a novel public key management scheme using the well-known web-of-trust or trust graph model. Our scheme is based on a binary tree formation of the network's nodes. The binary tree structure is proved very effective for building certificate chains between communicating nodes that are multihops away and the cumbersome problem of certificate chain discovery is avoided. We compare our scheme with related work and show that it presents several advantages, especially when a fair balancing between security and performance is desirable. Simulations of the proposed scheme under different scenarios demonstrate that it is effective in terms of tree formation, join and leave occurrences, and certificate chain establishment.

1. Introduction

Mobile Ad hoc Networks (MANETs) consist of a number of nodes which are typically unconfined and free to move and utilize wireless interfaces to communicate with each other without requiring any wireless infrastructure. The mobile nodes of such networks incorporate routing capabilities which give them the ability to construct multi-hop paths linking nodes which are not within their radio range. MANETs are currently employed in many areas of interest ranging from schools and universities to disaster places, while their use will be certainly widened in the near future. The self-organizing and dynamic nature of these networks however, makes the application of general-purpose protocols a real challenge for researchers. While many routing protocols have been proposed in the literature for MANETs, the establishment of a Public Key Infrastructure (PKI) in these networks has gathered little attention so far.

The absence of any fixed infrastructure and centralized authorities makes public key management for MANETs a very difficult task from both an algorithmic and computational point of view. Any public key-based security system must be provided by an efficient way of making the public key of a node available to another and proving in the same time the authenticity of this key. The solution to this problem in wired, general-purpose networks comes from the use of on-line or off-line servers that issue certificates to the nodes of the network. In MANETs, the absence of centralized services and the possible network partitions makes this problem hard to deal with. Recently, some public key management schemes for MANETs have been proposed. These schemes can be roughly classified into two categories. The first approach uses a set of nodes as servers that can provide partial certificates to a combiner by utilizing the concepts of threshold secret sharing [1–3]. Other server oriented solutions not utilizing threshold cryptography also exist like the Kaman one [4] which is based on Kerberos. The second category, which is our topic of interest, is based on the web-of-trust model [5, 6]. According to this approach, every node issues certificates only to the nodes it trusts. By considering every issued certificate as an edge, a certification graph is formed. If two nodes wish to exchange their public keys and form a common secret, they find a certification path in the graph and they can in this way authenticate each other. However, the major disadvantage of this approach is the cumbersome problem of finding a certification path in the graph. A solution to this problem is proposed in [7] where a virtual hierarchy is built among the nodes in the graph. Finally, an emerging third category attempts to combine trust graphs and threshold cryptography [8] in a fully distributed manner.

In this paper, we propose an approach similar to [9]. However, our scheme is based on a binary tree formation of the network's nodes rather on partitioning the network into clusters. Specifically, two alternative methods for binary tree formation are proposed each one having its pros and cons. By using this structure, the certificate path discovery problem is avoided and the place of each node in the tree can be easily found. Moreover, the frequent join and leave events in the network are efficiently dealt by modifying the tree structure where it is needed. In a nutshell, the proposed scheme has several advantages over other similar solutions, being more effective in terms of join and leave procedures, path discovery and certificate management, especially when security is not of top priority. Simulations of the proposed scheme under different scenarios show that it is affordable in terms of service times as well. On the other hand, when security is the prime focus, we offer a modified version of the proposed scheme which can deliver robust security services and effectively identify Denial of Service (DoS) attacks not addressed by similar mechanisms until now. Last but not least, we survey some methods for establishing and maintaining trust between the nodes of a MANET. Whilst this issue is very important, as it globally affects nodes' trustworthiness, it is not adequately addressed in the literature so far.

The rest of the paper is organized as follows. In the next section, we discuss how trust can be initially established and maintained between the nodes of a MANET. In Section 3, we present our binary tree-based protocol consisting of two alternative tree formation methods. The certificate chain discovery procedure is discussed in Section 4. Section 5 extensively surveys related work. Section 6 provides discussions and a theoretical comparison with other mechanisms that fall in the same category as our scheme. Simulation parameters and performance results are given in Section 7. Section 8 concludes the paper and gives some directions for further research.

Notation. An earlier and shorter conference version of this paper appeared in proceedings of IEEE ISWCS'08 [10]. This paper extends all sections of the aforementioned paper, addresses previous work thoroughly, provides detailed evaluation of the discussed scheme, and presents more rigorous arguments in some of our analysis.

2. Establishing and Maintaining Trust between Network Nodes

Most works in the field of public key management assume either that some sort of trust among network entities exists beforehand or that the nodes proceed with pairwise certification faithfully or at random. For instance, after certification, if a node is detected to behave aggressively or does not obey to the network rules then its certificate is revoked or left expired. Clearly, establishing trust among network nodes in a MANET remains a very challenging issue. Note that this issue is not the main focus of the current paper. However, it is considered important for every scheme that copes with trust in MANET, thus in this section, we address related work.

Usually, there is no external prior context at all among the participating entities. Bootstrapping from an existing infrastructure or exploiting proximity for expressing indexicality, as they are presented in [11], can furnish partial solutions towards solving this problem. For these reasons, trust and ad hoc networks can be thought in a sense as contradicting terms.

In many cases however, it is necessary to initialize security from the scratch for protecting subsequent interactions within the system. In this context, using Proofs of Work (PoW) in initialization of trust, and reputation systems can assist in establishing a certain degree of trustworthiness in the network [12, 13]. The objective behind PoW systems is that a verifier V can make sure that a prover P has successfully performed a certain computational task. The basic characteristic of a PoW system is that creating the proof must entail a predictable amount of work, while verifying the proof must be straightforward. Luo et al. [14] propose another method called URSA to secure network access control in a MANET. They propose a fully decentralized scheme to grant access control using certified tickets. Every prospective network member must acquire a ticket in order to grant network access. Tickets are generated by a subset of one or two-hops away neighbors that collaborate to enforce access control. Moreover, Saxena et al. [15] describe an admission control protocol based on bivariate polynomials that permits an ad hoc node to negotiate and validate a share-polynomial that can be used to establish a pairwise key between it and other nodes in a MANET. Of course, such schemes cannot fully guarantee that a node can be trusted but they assist in automatically exclude some of the malicious, misbehaving or selfish peers from joining the network.

When no centralized authority exists, as in the case of MANET, one popular method towards deriving positive conclusions whether a given node can be trusted is to employ a reputation rating [11, 16] or recommendation system [17]. The reputation ratings can be based on direct experience or recommendations by others in the network community or a combination of the two. Further down we describe in short the major proposals in the field; however a detailed analysis is beyond the scope of this paper. Work in [18], proposes the CONFIDANT protocol based on the Dynamic Source Routing (DSR) protocol. In CONFIDANT each node continuously monitors the behavior of its one hop neighbors. In order to isolate misbehaving nodes the noforwarding or other selfish behavior of any node is monitored and reported. The gathered information is forwarded to a reputation system that calibrates the rating of the nodes. The Watchdog and Pathrater mechanisms [19] are in fact two extensions to DSR. Watchdog relies on promiscuous mode operation of the ad hoc nodes with the aim of observing neighboring nodes behavior with respect to participation to basic network functions. Actually, the Watchdog mechanism comprises the basic assumption in any trust computational model. Pathrater exploits the knowledge from the Watchdog software module to select the most credible forwarding path for data packets. Other mechanisms award the unselfish nodes by giving them incentives.

Work in [20] proposes the Secure and Objective Reputation-based Incentive (SORI) method to promote packet forwarding and control egotistic actions. A unified network layer Token-based security solution for MANETs is given in [21]. According to this scheme, a node should hold a token in order to participate in any network operation while its adjacent nodes monitor any misbehavior in data packet forwarding functions. Another credit-based system for MANETs called Sprite is proposed in [22]. Sprite uses a credit system to provide incentives for nodes that collaborate and report actions truthfully. RFSTrust [17] proposes a trust model based on fuzzy recommendation similarity in order to quantify and evaluate the trustworthiness of nodes. RFSTrust includes five types of fuzzy trust recommendation relationships based on the fuzzy relation theory and a mathematical description for MANETs. Generally, while reputation systems work acceptably well in centralized realms their application in MANET scenarios require a decentralized reputation system, which in turn brings several issues in the foreground mostly related with the recommendations exchange system design and the avoidance of Sybil attacks.

Some other answers to the basic question "Who trust whom in a MANET and why?" do exist in terms of device authentication [23]. Yet, such solutions mandate in many cases some *a priori* configured trust relationship between the participating nodes. For example, every device joining the network can carry a device certificate proving its genuineness. Nevertheless, this requires a PKI infrastructure to sign all the certificates during the so-called network initialization phase. The same problem applies in the case of trusted computing oriented solutions. In our opinion establishing trust among network entities in a MANET remains very much an open research problem.

3. Proposed Solutions

In this section, we describe two similar solutions for building a binary tree of trust between the nodes of any MANET. The binary tree approach can greatly contribute to path discovery process optimization, and thus can facilitate the acquisition of certificate chain between the involved nodes. The first scheme starts from a single randomly chosen node, for example, the root of the tree and continues cascading until all willing-to-participate nodes join the tree. The other one hastens the formation of the binary tree by starting building tree branches from several different nodes simultaneously.

3.1. The Binary Tree-Based Scheme. The formation protocol starts when a given node, say $N0$ sends a HELLO message to its neighbours stating that it wants to initiate a tree-based trust relationship with them. Taking ad hoc on-demand distance vector (AODV) [24] as example, this is a Route Reply (RREP) with TTL = 1. For the moment, assume that there is no preestablished trust among all or some of the network nodes that comprise the MANET. So, the adjacent nodes may decide to accept the invitation or simply reject it. Accepting such an invitation from a given node means that the invited (peer) node is willing to proceed with a mutual-certification process with the initiator. The purpose of the protocol is to form a binary tree of trust between all network entities. Therefore, each node can provide certificates to a maximum of two neighboring nodes. All nodes have a {public, private} key pair created locally, so for every node pair each part signs the public key of the other using its private key and sends the result towards the other part. This tree forming procedure depicted in Figure 1 continues cascading requests from the root of the tree ($N0$) down to the leafs. Assuming that the network is dense enough the probability of having some—willing to participate—nodes (as $N12$ in Figure 1) left out of this process is negligible.

Figure 1(a) depicts the initial state of the network as well as each node's signal range. At some point, $N0$ initiates the protocol by sending pairwise-certification requests towards $N1$, $N2$ and $N3$ correspondingly. The latter nodes agree to participate, so they are pairwise-certified with $N0$. After that, they send pairwise-certification requests towards their neighbours, for example, $N3$ invites $N4$, $N5$ and $N6$. This situation is illustrated in Figure 1(b). The protocol continues until the binary tree depicted in Figure 1(c) is finally formed. When a given parent-node has completed the mutual-certification procedure with two child-nodes, it will drop any similar request coming from its neighbours. For example, in Figure 1(b) node $N0$ sends requests to $N1$, $N2$, and $N3$ but drops the reply from $N3$ since $N1$ and $N2$ have answered quicker to its request and have already been added in the binary tree. If a child-node has already been mutually-certified with a parent-node ignores post-dated pairwise certification requests send by others. To do so, each node must send in its HELLO messages its current state in the tree, that is, the bit 0 or 1 for non-members and members correspondingly. This is necessary in order to avoid redundant pairwise-certifications or loops between the leafs of the same tree. For instance, as $N6$ has already established a relationship with $N2$, drops the request originating from $N3$. It is worth noting that all nodes are supposed to be equal and the notation "parent" or "child" denotes their position in the tree. It is also stressed that for unsecured communications the nodes can use any possible available route. For example, if $N5$ is in the range of $N9$ they can exchange data directly.

However, to establish a secure relationship they must first obtain the certificate one another via the binary tree of trust, to setup a symmetric session key, and finally communicate directly as the case may be.

As already mentioned in Section 2 the main question of the certification procedure remains: "how can a node be convinced that a given public key, say K($N0$) truly belongs to node $N0$, so as to proceed with certification?" And, if certification succeeds, how one can rest assured that a certified node will continue to obey the network

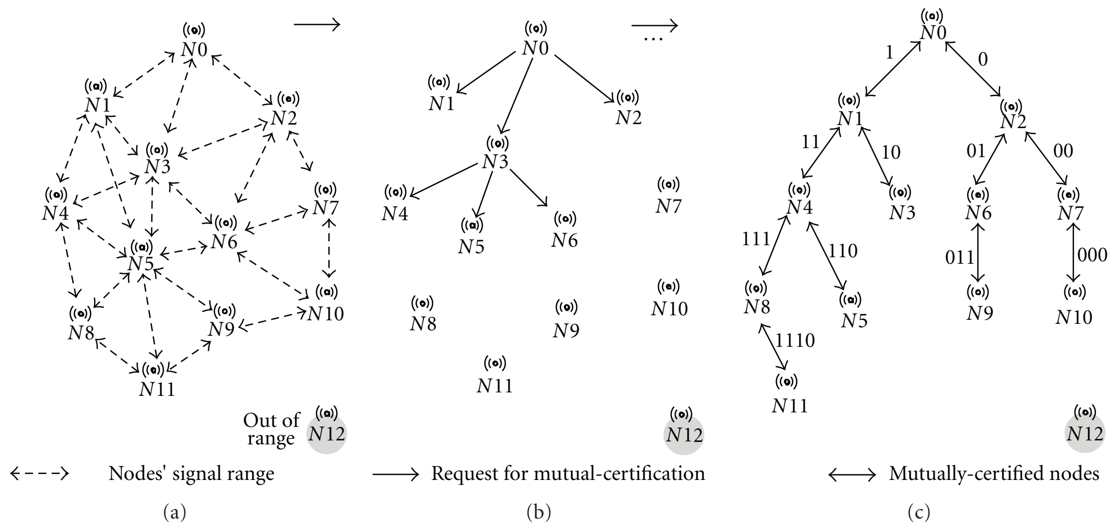


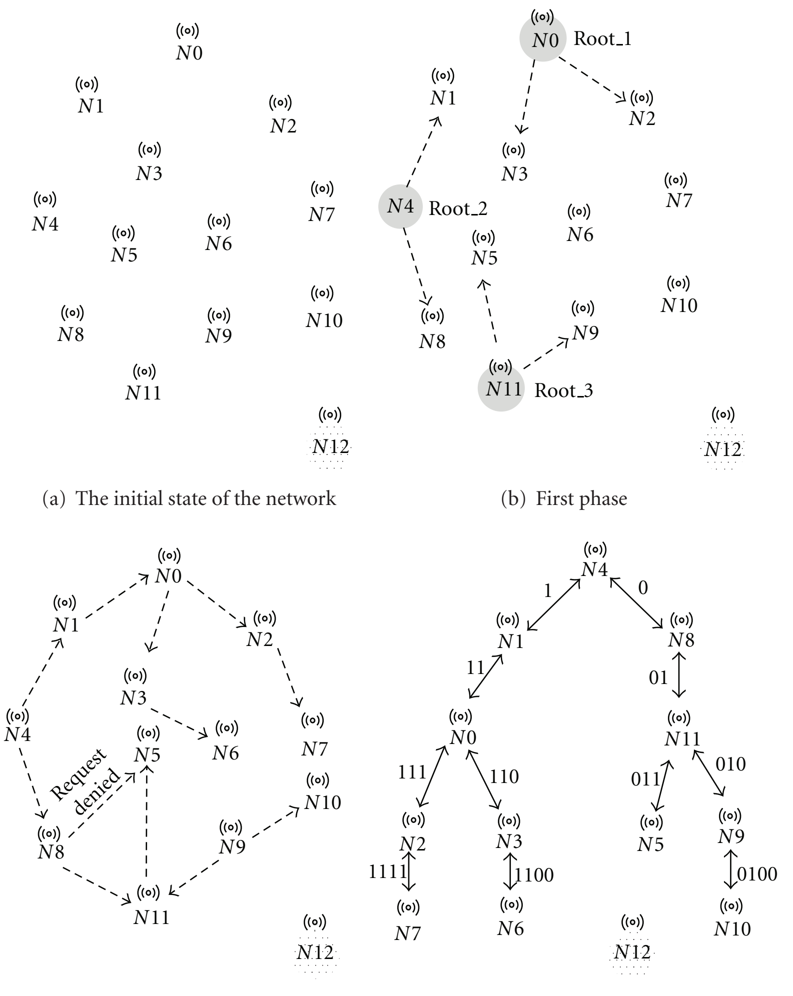
Figure 1: Formation of the binary tree of trust.

rules? Whilst all the approaches mentioned in Section 2 for initializing trust can be applied in our case, we assume a "friendship-based" and "commitment-driven" solution. That is, trustworthiness among nodes may depend on the existing friendship of users participating in the network. If two users, say Alice and Bob, are friends, they trust each other and their respective devices can exchange their public keys. There may be numerous reasons for a node Alice to believe that she is binding the Bob's public key to the real Bob and not Malloy. For example, nodes Alice and Bob may have exchanged their public keys via a side channel (e.g., over an infrared channel) at the time of a given physical encounter. As presented in [25], we can assume a bidirectional trustworthiness between two nodes, meaning that, if Alice trusts Bob, then Bob trusts Alice as well. This assumption stems from the statistical analysis of the Web of trust showing that about 2/3 of the links in the large strongly connected social network are bidirectional [26].

Overall, we can say that every node commits itself to the scheme; to be friendly, disciplinarian and behave legitimately. Otherwise, it will be isolated from the tree of trust and would not be able to send any packet via its neighbours. Also, to increase the level of trustworthiness, initially, every node may certify any other for a short period of time, say for a couple of hours. After that, if the "postulant" node proves good intentions, its certificate is renewed with a greater validity period. It is worth noting that detecting misbehaving nodes among one-hop nodes is quite easy due to the broadcast nature of wireless communications. The detection of misbehaving nodes can also be enforced using a reputation rating or recommendation system based on observation and/or second-hand information as discussed in Section 2. An already certified node must present a valid certificate to get a new one. Otherwise, the renewal procedure fails and implicitly the node is forced to disjoin from the network. Even though the proposed method imposes increased node overhead during its first stages, balances some time later after achieving a relative high degree of trust level between all the participants.

3.2. The Parallelized Binary Tree-Based Scheme. The binary tree-based scheme described in the previous subsection, can be easily parallelized in order to improve efficiency. Instead of launching the protocol from a given node, one can initiate the protocol by using simultaneously two or more nodes. The number of these nodes can be parameterized in the whole network. Every such root node leads to the construction of a small binary tree (which can be considered as a small cluster) and all these subtrees can be linked together through their root nodes forming a bigger network of trust. Linking different binary trees into one also implies that every node on each tree carries also the unique identity of the tree, that is, the IP address of the root.

Consider for example, the network in Figure 2(a). Suppose that nodes $N0$, $N4$, and $N11$ are randomly selected and they start the execution of the binary tree-based scheme. After, the first step of the protocol, three subtrees have been created (see Figure 2(b)). Every subtree should have a unique tree_ID, for example, the IP address of the root node. When a node receives an invitation from one of its neighbours, it should check whether this node has the same tree_ID. If the communicating nodes' tree_IDs are the same, then the invited node does not accept the invitation (otherwise a cycle would be formed). In the case that the tree_IDs are different, then both nodes agree randomly in one of the tree_IDs and inform all the other nodes in the two subtrees in order to all adopt the same tree_ID. For example, in Figure 2(c) node $N8$ has sent an invitation to node $N11$. The latter has accepted it, since $N8$ and $N11$ belong to different subtrees, and the rest of the nodes are notified that they belong now in the same binary tree. After that, if $N8$ sends an invitation to its other neighbour ($N5$), then this request will be denied since both nodes belong now in the same subtree.



(a) The initial state of the network (b) First phase

(c) Second phase (d) Finalization

Figure 2: Example of the parallelized binary tree-based scheme.

However, there is another one parameter that should be taken care of in order to guarantee that a binary tree is created. A node that has already accepted three invitations should not accept another one, even in the case that this request is coming from a different subtree node. If this restriction is not satisfied, then the formed tree would not be binary. When all nodes in the network have been visited (Figure 2(c)), a node having two adjacent edges should be chosen to be the root. For example, if node $N4$ is chosen in Figure 2(c) then the formed tree is the one depicted in Figure 2(d). Generally, this scheme performs faster when compared to those described in the previous subsection. However, this comes at a cost in complexity, that is, the merging process of different subtrees.

3.3. Handling Join and Leave. According to the proposed schemes the join and leave procedures are straightforward. Briefly, when a node leaves or an entrant joins the network only a branch of the tree is affected. More specifically, supposing that $N4$ in Figure 1(c) leaves the community, for example, moves out of range, nodes $N8$, $N5$ will seek parent in $N3$ or $N6$ depending on the topology and signal strength.

On the other hand, thinking of a scenario where $N12$ joins the network near the range of $N3$ it will establish a relationship with either $N3$, $N5$, $N6$, or $N9$. It is implied that in the rare case a newcomer cannot immediately find an association it must wait for some time until some other node roams out of that specific coverage area (a parent loses one child). In such occasions there is always the possibility for the node to roam to a new position until it finds a pair.

Lastly, the most complex leave situation is when the root node, say $N0$ in Figure 1(c), leaves the tree for some reason. Then $N1$ or $N2$, that is, the nodes closer to him, must replace $N0$. Assuming that $N1$ takes over the role of the root it must abandon $N3$, keep the connection with $N4$, and establish a direct relationship with $N2$. Consequently, $N3$ must seek for another parent. Even in this case, the join procedure is expected to complete after very few interactions, that is, new mutual-certifications between the corresponding nodes. A performance evaluation of join and leave for our scheme is given in Section 7.

4. Certificate Chain Discovery Procedure

For secure communication, any two nodes must be authenticated mutually. This means that each part must acquire and verify the certificate of the other. This can be fulfilled by constructing a certificate chain between them. In the following we consider an approach based on AODV [24]. However, our method can be embedded through proper extensions or slight modifications to any existing routing mechanism like Dynamic Source Routing (DSR), Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) and Cluster-based Routing Protocol (CBRP) to mention just a few.

AODV defines four message types which are Route Request (RREQs), Route Reply (RREPs), Route Error (RERRs), and Route Reply Acknowledgment (RREP-ACK). All message types are received via UDP, and normal IP header processing applies. According to AODV, every time a route to a new destination is needed, the node broadcasts a RREQ to discover it. Note that a route can be determined either when the RREQ finally reaches its intended destination, or an intermediate node that has a fresh route to the destination [24]. Upon that, the route is made available to the initiator of the RREQ by unicasting a RREP back to him. This is possible because each node receiving the initial request caches a route back to the originator. The binary structure further assists route discovery as each branch of the tree can be quickly identified by a binary sequence. For instance, referring to Figure 1(c) and starting from the root, the route to $N5$ is "110", where "1" means left and "0" right. This fact actually revokes the need for route optimization in every hop making the whole procedure particularly effective. Moreover, the length of the longest route in the binary tree will have on average $O(\log 2\, n)$ order, where n is the total number of nodes in the tree. This is easily concluded from [27] where it is shown that the mean value of the height of a randomly constructed binary tree with n nodes is equal to $a \ln n - b \ln \ln n + O(1)$ for constants $a = 4.311\cdots$ and $b = 1.953\cdots$.

Taking Figure 1(c), for example, in the following we describe the necessary steps for $N8$ (initiator) to build a certification chain with $N10$ (peer).

(a) To build the required certificate chain $N8$ needs to broadcast a special request message towards $N10$.This is necessary in order (i) to explicitly indicate to the peer that it asks for a certificate chain establishment, and (ii) to indicate to the intermediate nodes that they must forward the packet toward its final destination. AODV per se does not offer such message so we can consider at least two alternatives. First, insert the value "5" in the *Type* field of a standard RREQ message as shown in Figure 3(a); this field identifies the type of the message and is assigned values between 1 and 4 in AODV [24]. Second, use a special value, say "11", in the reserved field of a standard RREQ; this field is always sent as "0" in AODV and ignored on reception. In the following we choose to use the former option. Also, assume that the IP address of $N10$ is already known due to a previous RREQ.

(b) Upon reception of the RREQ message, $N10$ constructs the corresponding special type of RREP. Similarly to (a) it may insert the value "6" in the type field meaning that the packet must be treated as a certificate chain reply. Finally, $N10$ appends its own certificate to the message, signs the $\{\text{RREP}\|\text{Cert}(N10)\}$ block using its private key and appends it to the RREP. The format of the modified RREP packet is depicted in Figure 3(b). According to the local routing table, the resultant packet is sent back to $N8$ as a reply.

(c) All intermediate nodes must inspect every RREP that contains "6" in the type field. Specifically, $N7$ will validate the signature $\text{Sig}(\text{RREP}\|\text{Cert}(N10))_{N10}$ contained in the received message. Note, that $N7$ has the public key of $N10$, so it can securely verify the signature. Every attempt by means of a man-in-the-middle attack to alter the certificate of $N10$ or the original RREP will produce an error. Assuming that the signature check returns true, $N7$ will sign the $\{\text{RREP}||\text{Cert}(N10)\}$ using his own private key and forward the result along with the $\{\text{RREP}\|\text{Cert}(N10)\}$ block to $N2$. All the nodes in the path, that is, $N2$, $N0$, $N1$, $N4$ will repeat the same steps, as $N7$ did, until RREP reaches $N8$. If an error occurs at a given hop, that is the signature is not valid, the process is halted, and an RERR is generated and forwarded back to the initiator. We should mention that it is important for every node in the chain to digitally sign not only the original RREP but also the certificate of the target node ($N10$ in our case). This is done to prevent DoS attacks where the certificate of say $N10$ is altered when in transit by a man-in-the-middle attacker. So, $N8$ will receive a bogus certificate of $N10$. Later on, $N8$ and $N10$ having the certificates one another will try to establish a symmetric key by utilizing, for example, a challenge-response protocol. Naturally, the process will fail because the public key of $N10$ is invalid. This naturally leads to a DoS and consequently, the nodes must repeat the certificate chain discovery procedure from the ground up.

(d) As soon as the $\{\text{RREP}\|\text{Cert}(N10)\|\text{Sig}(\text{RREP}\|\text{Cert}(N10))_{N4}\}$ arrives to its final destination, $N8$ will check the validity of the signature using the local copy of $N4$'s public key. If everything is correct $N8$ will prepare his special type of RREP to be sent towards $N10$. The procedure is exactly the same as before but in the reverse order. This will allow $N10$ to successfully acquire the certificate of $N8$. Figure 4 provides an overview of all the aforementioned steps.

(e) After the two ends have acquired the certificate of each other they can agree on a per session shared secret (symmetric key) to communicate securely.

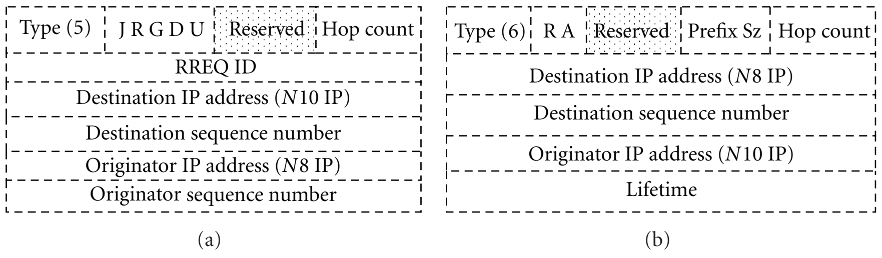
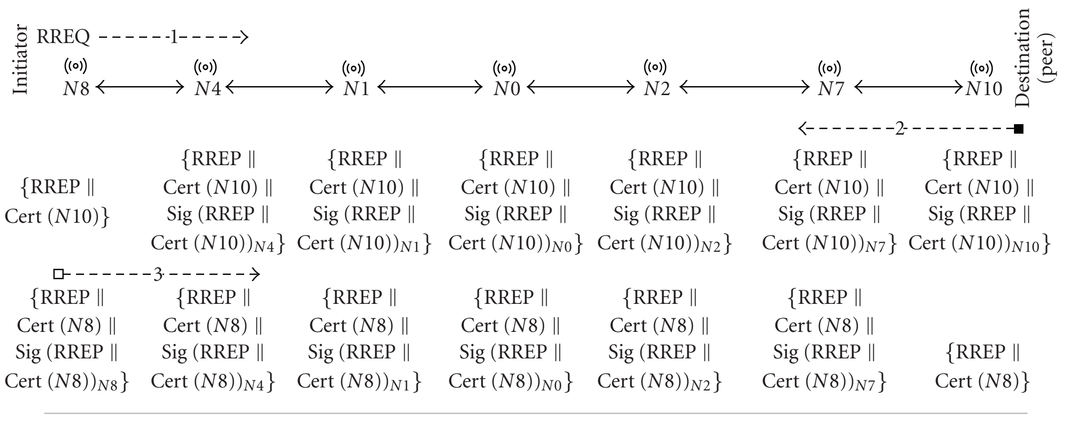

Type (5)
J R G D U
Reserved
Hop count
RREQ ID
Destination IP address (N10 IP)
Destination sequence number
Originator IP address (N8 IP)
Originator sequence number
(a)
Type (6)
R A
Reserved
Prefix Sz
Hop count
Destination IP address (N8 IP)
Destination sequence number
Originator IP address (N10 IP)
Lifetime
(b)

FIGURE 3: RREQ towards $N10$ and RREP towards $N8$.


Initiator
RREQ -1-
N8
N4
N1
N0
N2
N7
N10
Destination (peer)
-2-
{RREP || Cert (N10)}
{RREP || Cert (N10) || Sig (RREP || Cert (N10))N4}
{RREP || Cert (N10) || Sig (RREP || Cert (N10))N1}
{RREP || Cert (N10) || Sig (RREP || Cert (N10))N0}
{RREP || Cert (N10) || Sig (RREP || Cert (N10))N2}
{RREP || Cert (N10) || Sig (RREP || Cert (N10))N7}
{RREP || Cert (N10) || Sig (RREP || Cert (N10))N10}
-3-
{RREP || Cert (N8) || Sig (RREP || Cert (N8))N8}
{RREP || Cert (N8) || Sig (RREP || Cert (N8))N4}
{RREP || Cert (N8) || Sig (RREP || Cert (N8))N1}
{RREP || Cert (N8) || Sig (RREP || Cert (N8))N0}
{RREP || Cert (N8) || Sig (RREP || Cert (N8))N2}
{RREP || Cert (N8) || Sig (RREP || Cert (N8))N7}
{RREP || Cert (N8)}

where Sig $(x)_y$ is the signature of block x signed by the private key of node y and Cert (x) is the public key certificate of node x.

FIGURE 4: Certificate chain discovery procedure between $N8$ and $N10$.

5. Related Work

As already pointed out in the introduction, several mechanisms for self-organized public key management in MANETs have been proposed in the literature so far. In this section, we categorize them at a high level in partially centralized (usually found in hybrid MANETs) and infrastructureless or ad hoc ones. In a next section we compare our solution with similar typical schemes that subsume in the second category.

5.1. Partially Centralized Schemes. Works in the first category are based on some kind of Trusted Third Party (TTP), for example, special PKI servers, and Kerberos to issue and distribute digital certificates and public key pairs. The authors in [1] were the first to propose a partially distributed Certification Authority (CA) consisted of servers, combiners, and a dealer. Their scheme was based on threshold cryptography, that is, assuming that all partial signatures are collected a newcomer can compute the complete signature locally to acquire the public key certificate. In [3] the authors also propose a distributed CA based on threshold cryptography. They distribute the CA functionality over specially selected nodes based on the security and the physical characteristics of nodes. The selected nodes are called MObile Certificate Authority (MOCA) servers. Communication overheads are reduced by caching routes to MOCA servers. Work in [28] proposes the so called Distributed certification authority with probabilistic freshness for ad hoc networks (DICTATE) scheme. DICTATE is based on a combination of threshold and identity-based cryptosystems to guarantee the security, availability, and scalability of the certification function. To do so, it employs a hierarchical CA between one "mother" CA residing in the wired network, and a group of distributed CA in ad hoc network. Also, ensures that the distributed CA always processes a certificate update request in a finite amount of time and that an adversary cannot forge a certificate. Moreover, it guarantees that the same entity responds to a query request with the most recent version of the queried certificate in a certain probability.

The authors in [29] present IKM an ID-based Key Management system as a novel combination of ID-based and threshold cryptography. Actually, IKM is a certificateless solution in that public keys of mobile nodes are directly derivable from their known IDs plus some common information. Therefore, IKM eliminates the need for certificate-based authenticated public-key distribution indispensable in conventional public-key management schemes. Also, IKM features a novel construction method of ID-based public/private keys. Work in [30] proposes a solution of a distributed CA based on threshold cryptography as well. The authors derive their idea from the fact that in distributed CA when the number of neighbors of a node, also called node degree, reduces there is a substantial increase in the certification service delays. Their solution is based on the observation that when the node degree varies, the delays involved in obtaining certificates may be reduced by dynamically varying the threshold value. This can dynamically adjust the threshold value by monitoring the average node degree of the network and thereby prevent an increase in certification service delay. The authors in [31] suggest a CA cluster-based architecture by dividing the network into clusters. Eventually, each cluster head is able to maintain a CA information table, which contains a list of CA nodes in the local cluster as well as in the other ones. The distributed CA information is maintained among cluster heads, which reduces service response delay and the overall system overhead. Work in [32] proposes a system called mixed NETworks Trust infrastRUcture baSed on Threshold cryptography (NetTRUST). NetTRUST uses two CAs that ensure public key management, that is, central CAs in wired network, and mobile CAs in ad hoc network. Mobile CA servers emulate the CA role by using a (k, n) scheme, and the central CA servers delegate the CA role to mobile CA servers by using a (t, m) scheme. NetTRUST is decentralized, supports node mobility, and resists against mobile CA failures.

5.2. Infrastructureless Schemes. On the other hand, in infrastructureless or fully distributed solutions, like the one we propose here, security does not rely on TTPs or special kind of servers but on self-organization of nodes. Here, trust is usually propagated through a trust graph like the one employed by the well known PGP model. The authors of [33] propose a self-organized trust model for MANETs, in which trust among nodes is derived via physical contact. According to the authors every node issues public key certificates to other trustworthy nodes from its own domain. Nodes can authenticate each others using chains of trust. All nodes are assumed to store as many certificates as possible. In this model, trust establishment is originating from “offline trust relationships, which in turn are generated from general social relationships. A modified version of [33] is proposed in [34]. The authors introduce a boot server in order to initialize the system. This server calculates and distributes to each node a list of bindings (identifiers and public keys) and each of them generates the corresponding certificates. Upon that, a web-of-trust relationships is generated leading to a fully distributed system, that is, all nodes authenticate themselves via certificates chains. In [6] the authors describe an on-demand, fully localized, and hop-by-hop public key management scheme for MANETs. According to their method the network nodes self-generate public/private key pairs, issue certificates to any adjacent nodes, keep these certificates in their certificate repositories, and provide authentication service adaptively and quickly to the dynamic topology of the network without relying on any centralized entities. Another work that does not use threshold cryptography but assumes a self-organized PKI among the nodes of a MANET as proposed by [5] is given in [7]. The authors propose a protocol that establishes a virtual hierarchy in a peer-to-peer PKI. This protocol is suitable for the dynamic environments of MANETs since it is executed in a short time. Their protocol does not require to issue new certificates among PKI entities, facilitates the certification path discovery process and the maximum path length can be adapted to the characteristics of mobile terminals with limited processing and storage capacity. A very recent work that proposes a fully distributed public key certificate management system based on a hybrid approach, that is, in trust graphs and threshold cryptography, is given in [8]. The described solution permits users to issue public key certificates and to perform authentication via certificates' chains without the presence of any centralized management or trusted authorities. Due to the use of threshold cryptography the proposed system is able to resist against false public keys certification.

Work in [35] is also based on web-of-trust approach. Here, nodes act as CAs without the presence of any TTP. The system divides the network into clusters, such that nodes are assigned into different groups with unique identifiers. Authentication can be performed in a distributed manner, and newcomers are introduced by any trustable nodes of the same group. Nodes in the network monitor the behavior of each other and continuously update their trust tables. This endures the false certificate issued by dishonest users and malicious nodes, and avoids them to be selected as introducing nodes. The approach described in [36] is also based on distributed clustering and relies on trust values metric and behavior monitoring in order to distribute the role of certification authority (CA) in each cluster. It employs fully self-organized security and monitoring process to supervise behaviors of nodes with low trust level. Furthermore, the authors introduce a new concept to protect CAs and avoid the single point of failure in each cluster. Another Cluster-based approach is presented in [9]. The authors propose a practical model of public key certificate chain for MANET. Their scheme does not rely on a central server, but rather on Cluster-Based Routing Protocol (CBRP). CBRP is used to issue certificates in a distributed fashion. The certificates are chained effectively and the signed messages can be transferred over a certificate chain.

6. Comparison and Discussion

As already stated in the previous section, the focus of this paper is on fully distributed or decentralized schemes and in particular to those that are based on the web-of-trust model. So, to be able to compare our scheme with others we briefly categorize all solutions that carry that properties into localized schemes [6, 33], Cluster-based [9] and hierarchical ones [7]. In particular, the scheme in [7] builds a virtual hierarchy over an already fixed web-of-trust between the nodes in order to facilitate path discovery. Note, that here we only consider schemes that are directly comparable with ours. Hence, threshold cryptography schemes, hybrid approaches and the virtual hierarchy approach of [7] are left out. Naturally, our solution is also classified into the hierarchical category. According to localized schemes, each node must be mutually-certified with all its neighbours. As a result, the overhead for issuing, storing, and maintaining certificates is far larger when compared to our method. Moreover, join and leave procedures for localized mechanisms are generally more complex and require frequent interplay between many nodes of the network. According to our solution, each node is associated with maximum three other nodes and the join and leave procedures are straightforward, excluding that of which the root node leaves the tree (see Section 3.3).

Table 1: Comparison between schemes.

Properties	Our scheme	Localized [6]	Cluster-based [9]
Size of Replies (e.g., RREP)	Small	Large	Small
Scalable (certificate management)	Yes	No	Yes
Number of paths between two nodes	One	Multiple	Multiple
Path discovery	Easy	Hard	Mediocre
Resistance to DoS attacks	Yes	No	No
Join-leave	Easy	Hard	Mediocre

Another important issue is that localized schemes build one-way trust relationship, not mutual. Putting it another way, only the certificate of the peer is acquired by the initiator, not the opposite as our scheme mandates. However, in order for the two entities to become mutually authenticated each one of them must successfully obtain the public key certificate of the other. Moreover, according to localized schemes, the certificates of all the involved in the chain nodes are stacked all the way back to the initiator. Therefore, the more nodes in the certification path the bigger the RREP message towards the initiator will be. On the downside, our mechanism copes with this problem similarly to [9] which is a Cluster-based protocol. Specifically, each node in the path must locally authenticate any certification path message received from a previous node in the path. If this check fails then an error message is instantly sent towards the initiator. Furthermore, according to [9] each node in the path must send its certificate to the next mode for validating the RREP's signature. However, this is not necessary (on the contrary it creates extra overhead to each node) because every node retains a pairwise-certification relationship with the previous and next one in the path. So, every node uses its local copy of the certificate of the previous node to validate the signature. Actually, this is the safest way to do this.

To the best of our knowledge, localized schemes do not protect the integrity of the critical parts of the certificate chain messages by having each node signing them, thus they are prone to DoS attack scenarios. Another issue with Cluster-based oriented solutions like [9] is that only the RREP is integrity protected (signed); not the certificate of the peer node. This of course can lead to DoS attacks as already described in Section 4. Moreover, the certificate chain discovery procedure in [9], which as already mentioned is Cluster-based, requires route optimization in each hop, which is also avoided by our scheme; actually all routes are already optimized due to the (virtual) binary-tree-style topology. Table 1 compares the basic properties of each abovementioned scheme.

One possible problem with all the aforementioned solutions arises when a node in the path becomes compromised. Naturally, this issue is even harder, if not impossible, to tackle if the attacker is very powerful and owns a large number of nodes in the MANETs at some point. In such an event, a malicious node could falsify the certificate of the peer, construct as normal the $\text{Sig(RREP}\|\text{bogus\_Cert(peer))}_{\text{private\_key\_of\_malicious\_node\_}y}$ part and forward the message toward the next hop as normal. The next node in the chain is not able to detect the forgery, so the initiator will eventually receive a bogus certificate. This situation also leads to a DoS incident and breaks down the whole chain. Note, that no solution proposed so far deals with such an attack. Actually, similar scenarios can be repelled with a significant extra overhead. More analytically, each node in the path, after placing its own certificate, must resign the $\text{Sig(RREP}\|\text{Cert(peer))}_y$ part over again with its private key and also append each own certificate to the message before forwarding it to the next node.

For instance, referring to the example of Figure 4, the corresponding message for $N7$ to be sent towards $N2$ will be: $\{\text{RREP}\|\text{Cert}(N10)\|\text{Cert}(N7)\|\text{Sig}(\mathit{Sig}(\mathit{RREP}\|\mathit{Cert}(N10))_{N10}\|\text{Cert}(N7))_{N7}\}$. By doing so, the initiator ($N8$) will finally receive a reencapsulated signature from all nodes in the path as well as all nodes' certificates. Starting from the inner signature, $N8$ can sequentially recalculate all the signatures until the outer one. Actually, $N8$ is able to detect the point of failure and alert other nodes to exclude the misbehaving member. Although this procedure adds more overhead, has also a positive outcome. That is, each node acquires valid certificates of all other nodes in the chain. Storing the certificates until they become expired can accelerate future communications. In every case one must balance wisely between performance and security. So, if security is terminus then the aforementioned solution must be followed.

7. Performance Evaluation

The performance of the basic binary tree scheme described in Section 3.1 is simulated and the results are presented in this section.

7.1. General Considerations and Testbed Setup

7.1. General Considerations and Testbed Setup. In order to evaluate the proposed scheme we employ the well known NS/2 simulator [37]. However, we were forced to implement several additional routines either in TCL or C++ as follows.

(a) *create_tree*. The simulator employs this routine to build the binary tree. Tree construction begins from node $N0$. Using the procedure *check_neighbors* a given node discovers its adjacent nodes and sends them a message inviting them to join the tree. Note that initially, due to the ad hoc network topology, no node knows the status of its neighbors, so the only way to become aware is by sending them messages as described in Section 3.1. The first two nodes that will reply to the invitation will become the initiator node's children. All nodes that have already joined the tree do not answer to any invitation. This procedure is repeated for all network nodes.

(b) *check_neighbors*. Using this routine a node is able to discover its adjacent nodes. The procedure first checks the distance between two nodes and the signal strength to be able to decide if the nodes are in range.

(c) *send_tree_message*. This procedure creates two agents, one for the sending and one for the receiving node, and associates them. The C++ class *Agent/tree_message* manages the transmission and receiving of such messages.

(d) *recv_peer/recv_source*. These TCL procedures provide the appropriate functionality when a node dispatches a send_tree_message.

(e) *traverse_tree*. Used to display the final tree structure. To do so it traverses the tree and outputs the parent node and its children (if any).

(f) *lost_my_child*. It is called upon a leave event.

(g) *find_my_parent*. It is called either upon a join event (newcomer) or by a node that lost its parent and seeking for a new one in order to rejoin the tree.

(h) *add_binary_string*. Every time a given node joins the tree it acquires a binary label that designates its exact position in the tree. The label of the root node is null. This procedure is responsible to administer the label for each joining node.

(i) *start_secure_communication*. It implements the secure communication between two nodes (called the initiator and the peer). More specifically, when the initiator wants to securely communicate with a peer sends him its binary label in the tree. The peer sends its certificate along the tree towards the initiator.

(j) *use_certificate_chain*. It is used to discover the certificate path from a node called initiator to another node called peer.

The pseudocode for each one of the aforementioned routines is given in the appendix. The source code is also available upon request from the authors.

The simulation parameters are given in Table 2. We simulated 6 different scenarios in total by varying the number and the topology of nodes randomly. Specifically, we used 8, 20, 40, 60, 80, and 100 nodes for each scenario accordingly. A sample topology for the last 4 scenarios is depicted in Figure 5. The purpose of the evaluation is threefold: (a) to estimate the overall time to construct the binary-tree under different scenarios, (b) to estimate the overall time to reconstruct the tree after leave and join events, and (c) to estimate the network time required to build up a certification chain.

Table 2: Simulation parameters.

Parameter	*Meaning*
set val(chan) Channel/WirelessChannel;	# channel type
set val(prop) Propagation/TwoRayGround;	# radio-propagation model
set val(netif) Phy/WirelessPhy;	# network interface type
set val(mac) Mac/802_11;	# MAC type
set val(ifq) Queue/DropTail/PriQueue;	# interface queue type
set val(ll) LL;	# link layer type
set val(ant) Antenna/OmniAntenna;	# antenna model
set val(x) [depending on the scenario];	# X dimension of topology
set val(y) [depending on the scenario];	# Y dimension of topology
set val(cp) “”;	# node movement model
file set val(sc) “”;	# traffic model file
set val(ifqlen) 50;	# max packet in ifq
set val(nn) [depending on the scenario];	# number of mobile nodes
set val(seed) 0.0 set val(stop) 1000.0;	# simulation time
set val(rp) AODV;	# routing protocol

7.2. Results and Discussion. Figure 6 depicts the overall time needed to construct the binary tree depending on the scenario. The time seems to significantly increase when the number of nodes in the network grows; for example, from 20 to 60 one can spot an additional time penalty of 29 seconds. This is actually expected but it is not the only time-affecting parameter. The density (topology) of the nodes and the distance between them also affects the overall time to build up the binary tree structure. This can be inferred from the plot as the time of 29.7 seconds needed for 60 nodes unexpectedly diminishes to 17.2 when the number of nodes becomes 80. Therefore, it is projected that the time needed to construct the tree will be affected by two major factors; firstly the number of nodes participating in the network, and secondly the exact topology of the network.

The tree reconstruction times after multiple join and leave events happening nearly at the same time are given in Table 3 (see the fourth line of Table 3 in each scenario). We consider 3 different scenarios where a 10, 20, and 30% of randomly chosen tree nodes are involved in join or leave operations almost simultaneously. Each scenario considers 6 variations according to the number of nodes in the tree as the case may be. The recorded total times seem to be highly spanned in some cases, for example, for the 40 nodes case and the 20% variation the time is 0.26 seconds but for the 30% it climbs to 7.33. The same situation is perceived for several other scenarios, for example, for the 80 nodes case there is a large difference of approximately 12.5 seconds between the 10, 20% variations and the 30% one. On the other hand, we must consider the fact that these times are severely affected by the tree topology and the randomness of the scenario, that is, how many nodes leave/join the tree each time and the exact position of every node involved in a leave/join incident. These factors may affect one or several other nodes depending on the case. That is, if the root node leaves the tree then it is expected to affect several other nodes (see Section 3.3). Of course, this is not the case for a leaf node. Also note that the 30% scenario is not common even in highly dynamic MANETs, but it is considered here as an overstressing case.

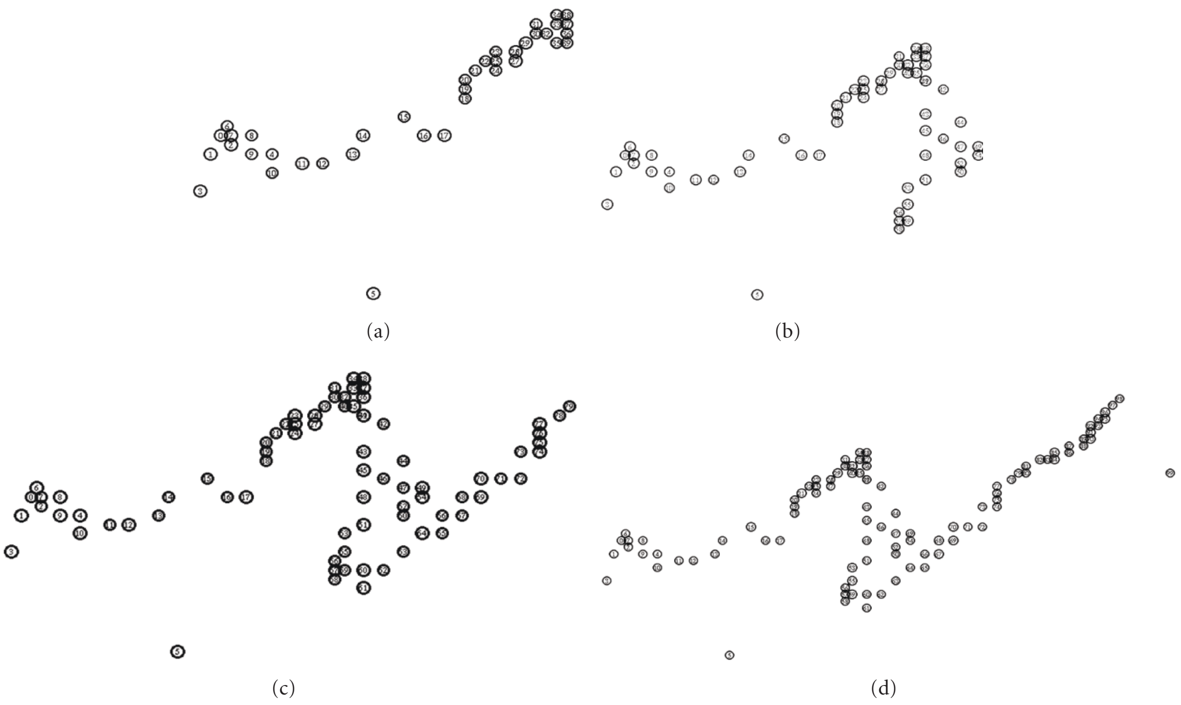


FIGURE 5: The simulation topology for 20 (a), 60 (b), 80 (c), and 100 (d) nodes.

TABLE 3: Timings and metrics for join/leave scenarios.

1	*Number of nodes in tree*		**8**			**20**	
2	*Percent of nodes engaged in join/leave operations*	**10%**	**20%**	**30%**	**10%**	**20%**	**30%**
3	*Number of messages sent*	9	9	14	3	4	13
4	*Total time to reconstruct the tree (seconds)*	0.08	0.06	0.12	0.03	0.06	0.20
5	*Mean time (milliseconds)*	8.4	7.0	8.4	10.6	14.2	15.3
6	*St. Deviation (milliseconds)*	2.4	1.9	3.7	1.5	6.8	11.7
7	*Confidence interval (80%)*	(7.4, 9.4)	(6.2, 7.8)	(7.1, 9.7)	(9.4, 11.7)	(9.9, 18.5)	(11.2, 19.5)
1	*Number of nodes in tree*		**40**			**60**	
2	*Percent of nodes engaged in join/leave operations*	**10%**	**20%**	**30%**	**10%**	**20%**	**30%**
3	*Number of messages sent*	24	28	40	33	58	49
4	*Total time to reconstructthe tree (seconds)*	0.28	0.26	7.33	8.05	7.85	1.56
5	*Mean time (milliseconds)*	11.5	9.3	183.3	244.1	135.3	31.9
6	*St. Deviation (milliseconds)*	5.6	3.5	710.4	904.2	630.1	144.4
7	*Confidence interval (80%)*	(10, 13)	(8.5, 10.2)	(39.3, 327.2)	(42.3, 445.8)	(29.3, 241.4)	(5.5, 58.3)
1	*Number of nodes in tree*		**80**			**100**	
2	*Percent of nodes engaged in join/leave operations*	**10%**	**20%**	**30%**	**10%**	**20%**	**30%**
3	*Number of messages sent*	41	69	70	60	89	89
4	*Total time to reconstructthe tree (seconds)*	0.53	0.84	13.07	3.37	5.25	11.12
5	*Mean time (milliseconds)*	13.0	12.2	186.8	56.2	59.0	124.9
6	*St. Deviation (milliseconds)*	8.3	7.6	1028.9	311.7	316.1	736.1
7	*Confidence interval (80%)*	(11.4, 14.7)	(11.1, 13.4)	(29.2, 344.4)	(4.6, 107.8)	(16.1, 101.9)	(24.9, 224.9)

The aforementioned situation of—in some cases—high deviations in the results is confirmed by the timings and associated metrics provided in Table 3. The third line of the table gives the total number of roundtrip messages required in each subscenario to reconstruct the tree. These messages were sent from all the affected nodes (i.e., nodes that lost their parent and/or others seeking to join the tree) in each sub-scenario in order for them to (re)join the binary tree. Note that an affected node may send one or more messages towards other nodes in an effort to eventually

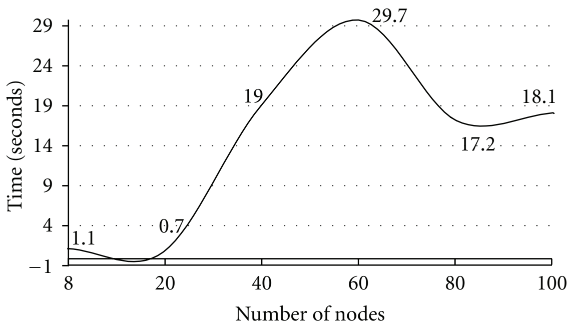


Figure 6: Binary tree construction timings.

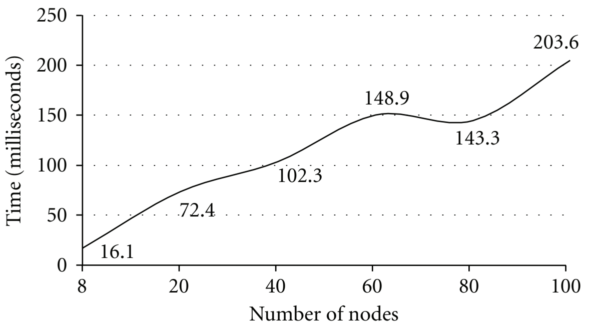


Figure 7: Certificate exchange timings (network only).

(re) join the tree. For instance, the 8 nodes case and the 10% variation as executed (1 node left) led to a total of 9 messages to be sent. These 9 messages required 0.08 seconds (75.5 milliseconds) altogether ending up to reforming the binary tree structure. For the 20 nodes scenario and the 10% variation only 3 messages were required and thus the overall time diminishes to 0.03 seconds (31.7 milliseconds). However, as the tree grows and the percentage of simultaneously join/leave incidents increases the number of messages required to reconstruct the tree tend—as a general rule—to augment. For example, the 100 nodes scenario with the 20, 30% variations required 89 messages to rebuild the binary tree, while the 10% variation triggered 60 messages from affected nodes. Also, message roundtrip times may differ significantly from sub-scenario to sub-scenario, and in some cases present unexpected peaks. This is justified by the unreliability of the wireless medium and channel allocation mechanisms (in our case these conditions are provided by NS/2). Besides, this situation is confirmed by the mean and standard deviation times given in the fifth and sixth line of Table 3 respectively. For instance, the 80 nodes scenario and the 30% variation produces a standard variation time of almost 1 second. Nevertheless, the spread of node message timings around the mean time is not certain to grow as the number of nodes increases. Therefore, we provide an 80 percent confidence interval metric to provide a better estimation of the range within which the true value of the statistic parameters of Table 3 lie. Generally, one can say that these timings heavily depend on the randomness of each scenario and the topology of the tree as noted again earlier.

Figure 7 presents the overall time needed for two nodes to exchange their certificates as described in Section 4. More specifically, from an edge leaf node to another; taking Figure 1 as an example from $N11$ to $N10$. This time is also proportional to the number of nodes in the network, that is, the depth of tree. Here, we only consider the network time to fulfill such a request for the worst case scenario. This is justified by the fact that the cryptographic functions (see Section 4) involved heavily rely on the device employed as the case may be. This gives one the advantage to easily recalculate the total times when changing the hardware characteristics of a device or the security algorithm employed. For instance, when speaking for modern processors, for example, an Intel Core 2 1.83 GHz processor under Windows Vista in 32-bit mode an RSA 1024 plain signature/verification requires 1.48 and 0.07 milliseconds, respectively, (see http://www.cryptopp.com/benchmarks.html). For a rather obsolete mobile device like the Sharp Zaurus SL-5500G (206MHz Intel SA-1110 StrongARM) the RSA 1024 plain generation/verification time is 78.0 and 4.3 milliseconds, respectively, [38]. Additional signature generation timings for different hardware settings and key lengths can be found in [39]. In a nutshell we can argue that our scheme requires few hundreds of milliseconds to build up a certificate chain between two nodes. This is because the network time is minimized due to the binary-tree structure (i.e., for 80 nodes it only requires 143 ms), while the public key operations take up the biggest portion requiring two signature operations per node except the initiator and the peer. However, in all cases, the signature verification time can be considered negligible. Overall, the binary tree formation procedure and maintenance may be time consuming but it really compensates when certificate exchange comes into play.

8. Conclusions

It is common ground that a MANET environment must be self-organizing and highly adaptive. When focusing on security, trust relationship between nodes in such network dynamically changes due to several causes including temporary problem, join/leave occurrence, hostile or selfish behavior, and so forth. The high-level contribution of this work is to propose and evaluate a public key management scheme based on a binary tree formation of the network's nodes. The motivation here is that the binary tree structure is able to boost any certificate chaining transaction and accelerate the re-assemble of the "trust graph" after join/leave events. We discuss and analyze two versions of the same method for building and maintaining a binary tree of trust between the nodes of a MANET. The first one triggers the formation procedure from a single randomly selected node, while the other hastens the creation of the binary tree by starting concurrently from several different nodes. We consider related work to a great extend and theoretically compare our proposal with schemes that classify in the same category. While our analysis does not concentrate on how the nodes acquire an initial level of trust and how they maintain it we survey the literature for solutions to this problem. Evaluation facts of the proposed scheme under different scenarios via simulations are also provided. Results show that the binary tree (re)structure is attainable and affordable in terms of service times while at the same time achieves optimal performance in case of certificate chain establishment. As a statement of direction, we are currently working on refining and simulating the parallelized binary tree-based scheme in order to obtain a better view on the performance of our schemes.

Appendix

A. Pseudocode of NS/2 Custom Routines

A.1. Procedure create_tree

```
create_tree()
# Creates the whole tree structure of the ad hoc network
# for each node find neighbors node and send them a message
to ask if they can be a child
for each node i {
    for each node j, i ≠ j {
        call procedure check_neighbors(node i, node j)
        if node i and node j are neighbors
            call procedure send_tree_message(i, j, "child")
    }
}
```

A.2. Procedure check_neighbors

```
check_neighbors(node a, node b)
# Check if node a is neighbor of node b
compute the distance between node a and node b
if distance <= antenna range
    return a is neighbor of b
else
    return a is not neighbor of b
```

A.3. Procedure send_tree_message

```
send_tree_message(source_node, peer_node, message_
type)
# Send a message from source to peer to invite him as a child

Create Agent/tree_message for source_node
Create Agent/tree_message for peer_node

Connect the agents

if {$message_type = = "child"} {
    # the source_node invites peer_node to be his child
    call tree_messageAgent("child")
} else {
    # the source_node invites peer_node to be his parent
    call tree_messageAgent("search_parent")
    }
}
```

A.4. Procedures used by a node to become parent or child node.

a. Procedure Recv_Peer_Parent

```
recv_peer_parent(source_node)
# This function is executed when a node receive a request to be
parent

if peer_node is not in the same subtree of source_node {
    if peer_node has not left child {
        set source_node as peer_node's left child
    }
    else if peer_node has not right child {
        set source_node as peer_node's right child
    }
    else {
        # cannot be a parent
        return failure
    }
}
```

b. Procedure Recv_Source_Parent

```
recv_source_parent(peer_node)
# This function is executed when a node receives the answer of
a candidate parent

if peer_node accepts to be a parent {
    set peer_node as parent of source_node
    # add binary label to specify the position of the node in the
tree
    call add_binary_string(source)
}
```

c. Procedure Recv_Peer_Child

```
recv_peer_child(source_node)
# This function is executed when a node receives a message to
be a child

if peer_node has not a parent {
    # Accept to be child
    set peer_node as child of source_node
    return success
}
else {
    return failure
}
```

d. Procedure Recv_Source_Child

```
recv_source_child(return, peer_node)
# This function is executed when a node receives the answer of
a candidate child

if return is success {
    if source_node has not left child {
        set peer_node as source_node's left child
        set binary_label "1"
}
else if source_node has not right child {
      set peer_node as source_node's right child
      set binary_label "0"
   }
}
```

A.5. Procedure traverse_tree

```
traverse_tree()
# This function traverses the tree. For each node display the parent,
# the children and the binary label.

for each node i {
   print parent node
   if node i has left child
      print left child
   if node i has right child
      print right child
   print node's binary label
}
```

A.6. Procedure Lost_My_Child

```
lost_my_child()
# Remove children nodes that are not neighbors anymore

for each node i {
   if node i has left child {
      call procedure check_neighbors(node i, left child)
      if node i and left child are not neighbors {
         set node i's left child as null
      }
   }

   if node i has right child {
      call procedure check_neighbors(node i, right child)
      if node i and right child are not neighbors {
         set node i's right child as null
      }
   }
}
```

A.7. Procedure find_my_parent

```
find_my_parent()
# If a new node joins the tree or an existing node is not anymore
# neighbour with his parent, look for a new parent (recreate the tree structure)

for each node i {

   # if the current node has lost his parent or a new node is coming
   call procedure check_neighbors(node i, parent)
   if (node i and his parent are not neighbors) or (node i has not parent) {
      for each node j {
         if i ≠ j {
            call procedure check_neighbors(node i, node j)
            if (node i and node j are neighbors) {
               call procedure send_tree_message(i, j, "parent")
            }
            }
         }
      }
}
```

A.8. Procedure add_binary_string

```
add_binary_string(node_id)
# Define a binary string for each node in relation with its position
# in the tree. Left child of root node has 1 and right child of root node has 0

# the node is leaf
If node_id = "null" {
   return
}

set parent_binary_label as the binary_label of node_id's parent
if node_id is left child {
   set node_id's binary_label as parent_binary_label + "1"
}
else {
   set node_id's binary_label as parent_binary_label + "0"
}

# Recursively call
call procedure add_binary_string(node_id's left child)
call procedure add_binary_string(node_id's right child)
```

A.9. Procedure start_secure_communication

```
start_secure_communication(source_node, peer_node)
# A node send its IP address to another node in order to communicate safely using certificate chain

call procedure use_certificate_chain(source_node, binary_label of peer_node)
```

A.10. 10. Procedure use_certificate_chain

```
use_certificate_chain(source_node,
peer_node_binary_label)
# Discover the certificate path from a node to another node
# Decide the next node in the path according to the binary string
(position) of the peer node

call procedure where_to_give_certificate(source_node, peer_node_binary_label)

if (next_path="parent") {
    call procedure send_certificate_message(source_node, parent)
}
else if (next_path="left child") {
    call procedure send_certificate_message(source_node, left child)
}
else
call procedure send_certificate_message(source_node, right child)
}
```

References

[1] L. Zhou and Z. J. Haas, "Securing ad hoc networks," *IEEE Network*, vol. 13, no. 6, pp. 24–30, 1999.

[2] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang, "Providing robust and ubiquitous security support for mobile ad-hoc networks," in *Proceedings of the International Conference on Network Protocols (ICNP '01)*, pp. 251–260, November 2001.

[3] S. Yi and R. Kravets, "MOCA: mobile certificate authority for wireless ad-hoc networks," in *Proceedings of the 2nd Annual PKI Research Workshop (PKI '03)*, 2003.

[4] A. Pirzada and C. McDonald, "Kerberos assisted authentication in mobile ad-hoc networks," in *Proceedings of the 27th Australasian Computer Science Conference*, vol. 26, pp. 41–46, 2004.

[5] J.-P. Hubaux, L. Buttyán, and S. Čapkun, "The quest for security in mobile ad hoc networks," in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '01)*, pp. 146–155, ACM, 2001.

[6] R. Li, J. Li, P. Liu, and H.-H. Chen, "On-demand public-key management for mobile ad hoc networks," *Wireless Communications and Mobile Computing*, vol. 6, no. 3, pp. 295–306, 2006.

[7] C. Satizábal, J. Hernández-Serrano, J. Forné, and J. Pegueroles, "Building a virtual hierarchy to simplify certification path discovery in mobile ad-hoc networks," *Computer Communications*, vol. 30, no. 7, pp. 1498–1512, 2007.

[8] M. Omar, Y. Challal, and A. Bouabdallah, "Reliable and fully distributed trust model for mobile ad hoc networks," *Computers and Security*, vol. 28, no. 3-4, pp. 199–214, 2009.

[9] G. Hahn, T. Kwon, S. Kim, and J. Song, "Cluster-based certificate chain for mobile ad hoc networks," in *Proceedings of the International Conference on Computational Science and Applications (ICCSA '10)*, vol. 3981 of *Lecture Notes in Computer Science*, pp. 769–778, Springer, 2006.

[10] G. Kambourakis, E. Konstantinou, and S. Gritzalis, "Binary tree based public-key management for mobile ad hoc networks," in *Proceedings of the IEEE International Symposium on Wireless Communication Systems (ISWCS '08)*, pp. 687–692, IEEE, Reykjavik, Iceland, October 2008.

[11] N. Asokan and L. Tarkkala, "Issues in initializing security," in *Proceedings of the 5th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT '05)*, pp. 460–465, IEEE, 2005.

[12] C. Dwork and M. Naor, "Pricing via processing or combating junk mail," in *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '92)*, vol. 740 of *Lecture Notes in Computer Science*, pp. 139–147, Springer, 1992.

[13] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, "Moderately hard, memory-bound functions," *ACM Transactions on Internet Technology*, vol. 5, no. 2, pp. 299–327, 2005.

[14] H. Luo, J. Kong, P. Zerfos, S. Lu, and L. Zhang, "URSA: ubiquitous and robust access control for mobile ad hoc networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 6, pp. 1049–1063, 2004.

[15] N. Saxena, G. Tsudik, and J. H. Yi, "Efficient node admission for short-lived mobile ad hoc networks," in *Proceedings of the 13th IEEE International Conference on Network Protocols (ICNP '05)*, pp. 269–278, Boston, Mass, USA, November 2005.

[16] G. F. Marias, D. Flitzanis, K. Mandalas, and P. Georgiadis, "Cooperation enforcement schemes for MANETs: a survey," *Jouranl of Wireless Communications and Mobile Computing*, vol. 6, no. 3, pp. 319–332, 2006.

[17] J. Luo, X. Liu, and M. Fan, "A trust model based on fuzzy recommendation for mobile ad-hoc networks," *Computer Networks*, vol. 53, no. 14, pp. 2396–2407, 2009.

[18] S. Buchegger and J.-Y. L. Le Boudec, "Performance analysis of the confidant protocol," in *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC '02)*, pp. 226–236, ACM, New York, NY, USA, 2002.

[19] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM '00)*, pp. 255–265, IEEE, New York, NY, USA, 2000.

[20] Q. He, D. Wu, and P. Khosla, "SORI: a secure and objective reputation-based incentive scheme for ad-hoc networks," in *Proceedings of the IEEE Wireless Communications & Networking Conference (WCNC '04)*, vol. 2, pp. 825–830, March 2004.

[21] H. Yang, J. Shu, X. Meng, and S. Lu, "SCAN: self-organized network-layer security in mobile ad hoc networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 261–273, 2006.

[22] S. Zhong, J. Chen, and Y. R. Yang, "Sprite: a simple, cheat-proof, credit-based system for mobile ad-hoc networks," in *Proceedings of the IEEE Communications Society Conference on Computer Communications (INFOCOM '03)*, vol. 3, pp. 1987–1997, IEEE, 2003.

[23] G. Kambourakis, S. Gritzalis, and J.-H. Park, "Device authentication in wireless and pervasive environments," *Intelligent Automation and Soft Computing*, vol. 16, no. 3, pp. 399–418, 2010.

[24] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," IETF RFC 3561, July 2003.

[25] C. Zhang, Y. Song, and Y. Fang, "Modeling secure connectivity of self-organized wireless ad hoc networks," in *Proceedings of the 27th IEEE Communications Society Conference on Computer Communications (INFOCOM '08)*, pp. 825–833, April 2008.

[26] "Keyanalyze—analysis of a large OpenPGP ring," analysis by Drew Streib, 2009.

[27] B. Reed, "The height of a random binary search tree," *Journal of the ACM*, vol. 50, no. 3, pp. 306–332, 2003.

[28] J. Luo, J.-P. Hubaux, and P. T. Eugster, "DICTATE: distributed certification authority with probabilistic freshness for ad hoc networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 4, pp. 311–323, 2005.

[29] Y. Zhang, W. Liu, W. Lou, and Y. Fang, "Securing mobile ad hoc networks with certificateless public keys," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 4, pp. 386–399, 2006.

[30] S. Raghani, D. Toshniwal, and R. Joshi, "Dynamic support for distributed certification authority in mobile ad hoc networks," in *Proceedings of the International Conference on Hybrid Information Technology (ICHIT '06)*, pp. 424–432, November 2006.

[31] Y. Dong, A.-F. Sui, S. M. Yiu, V. O. K. Li, and L. C. K. Hui, "Providing distributed certificate authority service in cluster-based mobile ad hoc networks," *Computer Communications*, vol. 30, no. 11-12, pp. 2442–2452, 2007.

[32] M. Omar, Y. Challal, and A. Bouabdallah, "NetTRUST: mixed networks trust infrastructure based on threshold cryptography," in *Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks (SecureComm '07)*, pp. 2–10, 2007.

[33] S. Capkun, L. Buttyan, and J.-P. Hubaux, "Self-organized public-key management for mobile ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 2, no. 1, pp. 52–64, 2003.

[34] K. Ren, T. Li, Z. Wan, F. Bao, R. H. Deng, and K. Kim, "Highly reliable trust establishment scheme in ad hoc networks," *Computer Networks*, vol. 45, no. 6, pp. 687–699, 2004.

[35] E. C. H. Ngai and M. R. Lyu, "Trust- and clustering-based authentication services in mobile ad hoc networks," in *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops*, pp. 582–587, 2004.

[36] A. Rachedi and A. Benslimane, "Trust and mobility-based clustering algorithm for secure mobile ad hoc networks," in *Proceedings of the 2nd International Conference on Systems and Networks Communications (ICSNC '06)*, pp. 72–78, IEEE, October 2006.

[37] "NS-2, Network Simulator 2," http://www.isi.edu/nsnam/ns/.

[38] D. Westhoff, B. Lamparter, C. Paar, and A. Weimerskirch, "On digital signatures in ad hoc networks," *European Transactions on Telecommunications*, vol. 16, no. 5, pp. 411–425, 2005.

[39] X. Ding, D. Mazzocchi, and G. Tsudik, "Equipping smart devices with public key signatures," *ACM Transactions on Internet Technology*, vol. 7, no. 1, pp. 1–3, 2007.