Securing the OLSR protocol

Cedric Adjih, Thomas Clausen, Philippe Jacquet, Anis Laouiti, Paul Mühlethaler, Daniele Raffo

INRIA Rocquencourt, Projet Hipercom,
Domaine de Voluceau, B.P.105, 78153 Le Chesnay cedex, France
Telephone: +33 1 3963 5363 Fax: +33 1 3963 5363
Email: {Cedric.Adjih,Thomas.Clausen,Philippe.Jacquet,Anis.Laouiti,Paul.Muhlethaler,Daniele.Raffo}@inria.fr

***Abstract*—**

In this paper, we examine security issues related to proactive routing protocols for MANETs. Specifically, we investigate security properties of the Optimized Link-State Routing Protocol - one example of a proactive routing protocol for MANETs. We investigate the possible attacks against the integrity of the network routing infrastructure, and present techniques for countering a variety of such attacks. Our main approach is based on authentication checks of information injected into the network. However even with perfect authentication check, replay attacks are still possible. Hence, we develop a distributed timestamp-based approach for verifying if a message is "old" or "current". We finally present two different, simple algorithms for distributing public keys in a MANET, in order to provide a mechanism permitting authentication checks to be conducted. While we use OLSR as an example protocol for our studies, we argue that the presented techniques apply equally to any proactive routing protocol for MANETs

I. Introduction

A Mobile Ad-hoc NETwork (MANET) is a collection of nodes which are able to connect on a wireless medium forming an arbitrary and dynamic network. Implicitly herein is the ability for the network topology to change over time as links in the network appear and disappear.

In order to enable communication between any two nodes in such a MANET, a routing protocol is employed. The abstract task of the routing protocol is to discover the topology (and, as the the network is dynamic, continuing changes to the topology) to ensure that each node is able to acquire a recent image of the network topology for constructing routes.

Currently, two complimentary classes of routing protocols exist in the MANET world. Reactive protocols acquire routes on demand through flooding a "route request" (which typically also records the path taken) and receiving a "route reply" (typically signaling the path taken by the route request to arrive at the destination node). I.e. the required parts of the topology graph is constructed in a node only when needed for data traffic communication. Reactive MANET routing protocols include AODV [13] and DSR [9].

The other class of MANET routing protocols is proactive, i.e. the routing protocol ensures that all nodes at all times have sufficient topological information to construct routes to all destinations in the network. This is achieved through periodic message exchange. Proactive MANET routing protocols include OLSR [1] and TBRPF [12]

A. Security Issues

A significant issue in the ad-hoc domain, is that of the integrity of the network itself. AODV, DSR, OLSR and TBRPF allow, according to their specifications, any node to participate in the network - the assumption being that all nodes are well-behaving and welcome. If that assumptions fails - that the network may be subject to malicious nodes - the integrity of the network fails.

An orthogonal security issue is that of maintaining confidentiality and integrity of the data being exchanged between communications endpoints in the network (e.g. between a mail server and a mail client). The task of ensuring end-to-end security of data communications in MANETs is equivalent to that of securing end-to-end security in traditional wire-line networks, and is not considered further in this paper.

The primary issue with respect to securing MANET routing protocols is thus that of ensuring the network integrity, even in presence of malicious nodes. Security extensions to the reactive protocols AODV and DSR exist, in form of SAODV [7] and Ariadne [8]. Assuming that a mechanism for key distribution is in place, these extensions employ digital signatures on the route request and route reply messages. The basic principle being, that each node verifies the signature of a message and - if valid - modifies the message (if applicable), signs it itself and forwards the message.

In this paper, we will investigate the issues of security in proactive MANET routing protocols, especially with emphasis on providing a security extension to OLSR.

B. Paper outline

The remainder of this paper is thus organized as follows: section II presents OLSR with sufficient details to device security mechanisms which will integrate with the protocol. Following, section III will describe the vulnerabilities of proactive routing protocols, utilizing OLSR for exemplifying the threats to which any proactive ad-hoc routing protocol is vulnerable. Section IV will proceed by describing a basic mechanism for securing OLSR. This section assumes that nodes are either untrusted or trusted - and that trusted nodes are not compromised. The mechanism proposed in this section also implies the existence of "global timestamps" and assumes that there exists a mechanism for distributing cryptographic keys between the nodes in the network. Thus, section V explores the problem of providing the nodes in the network with a suitable way of comparing

"timestamps" to determine if messages are "too old". This provides a way of dealing with the problem of message replays, in which a node repeats "old" messages which were previously transmitted (and correctly signed). Following, section VI describes mechanisms for distributing cryptographic keys in a MANET. This section assumes that asymmetric public-key cryptography is employed, and presents two simple solutions for key distribution. Finally, the paper is concluded in section VII, with a discussion of the applicability of the proposed security architecture.

II. The Optimized Link State Routing Protocol

The Optimized Link State Routing protocol (OLSR) [3], [1] is a proactive link state routing protocol, designed specifically for mobile ad-hoc networks. OLSR employs an optimized flooding mechanism for diffusing link-state information, and diffuses only partial link-state to all nodes in the network.

In this section, we will describe the elements of OLSR, required for the purpose of investigating security issues. A complete description of OLSR can be found in [1].

A. OLSR Control Traffic

Control traffic in OLSR is exchanged through two different types of messages: "HELLO" and "TC" messages. HELLO messages are exchanged periodically among neighbor nodes, in order to detect links to neighbors, to detect the identity of neighbors and to signal MPR selection. TC messages are periodically flooded to the entire network, in order to signal link-state information to all nodes.

1) HELLO messages: HELLO messages are emitted periodically by a node, including its own address as well as encoding three lists: a list of neighbors, from which control traffic has been heard (but where bi-directionality is not yet confirmed), a list of neighbor nodes, with which bidirectional communication has been established, and a list of neighbor nodes, which have been selected to act as MPR for the originator of the HELLO message. HELLO messages are exchanged between neighbor nodes only.

Upon receiving a HELLO message, a node examines the lists of addresses. If its own address is included, it is confirmed that bi-directional communication is possible between the originator and the recipient of the HELLO message. When a link is confirmed as bi-directional, this is advertised periodically by a node with a corresponding link status of "symmetric".

In addition to information about neighbor nodes, periodic exchange of HELLO messages allows each node to maintain information describing the links between neighbor nodes and nodes which are two hops away. This information is recorded in a nodes 2-hop neighbor set and is explicitly utilized for the MPR optimization - the core optimization of OLSR, described in section II-A.3.

2) TC messages: Like HELLO messages, TC messages are emitted periodically by a node. The purpose of a TC message is to diffuse topological information to the entire network. Thus, a TC message contains a set of bi-directional links between a node and a subset of its neighbors. For a discussion on the selection of which neighbors to include in the TC messages to provide sufficient topology information, refer to [1] and [4]. TC messages are diffused to the entire network, employing the MPR optimization described in section II-A.3

3) Multipoint Relay Selection and Signaling: The core optimization in OLSR is that of Multipoint Relays (MPRs). The concept is as follows: each node must select MPRs from among its neighbor nodes such that a message emitted by a node and repeated by the MPR nodes will be received by all nodes two hops away. MPR selection is performed based on the 2-hop neighbor set received through the exchange of HELLO messages, and is signaled through the same mechanism: a link-status of "MPR" specifies that the link between the originator of the HELLO message and the listed address is symmetric - and that the node with the included address is selected as MPR by the originator.

Thus, each node maintains an *MPR selector set*, describing the set of nodes which have selected it as MPR. Upon receiving an OLSR control message, a node will consult the MPR selector set for determining if the message is to be retransmitted: if the last-hop of the control message is an MPR selector, then the message is to be retransmitted - otherwise it is not retransmitted. Figure 1 shows a node with neighbors and 2-hop neighbors. In order to achieve a network-wide broadcast, it suffices that a broadcast transmission be repeated by a subset of the neighbors. This subset is made up from the MPR-set of the node. For further information, including an efficient heuristic for computing the MPR set of a node, refer to [14].

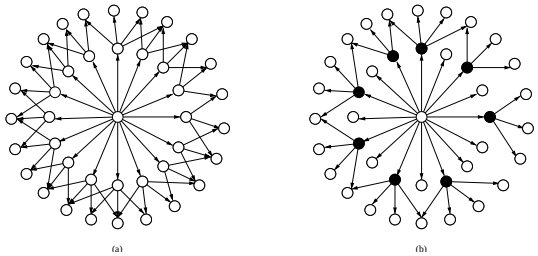

Fig. 1. Two hop neighbors and "multipoint relays" (the solid circles) of a node. (a) illustrates the situation where all neighbors retransmit a broadcast, (b) illustrates where only the MPRs of a node retransmit the broadcast

B. OLSR Message Format and Packets

OLSR control messages are communicated using a "transport protocol" defined by a general packet format containing individual control messages, as well as rules governing the processing of such packets and messages. In this section, we will outline this transport protocol. The purpose is to outline how security extensions can be almost effortlessly included, and to understand the mechanisms under which the security extensions must be designed.

The OLSR packet format is given in figure 2.

It is important to notice, that while messages potentially may be intended to be broadcasted to the entire network (e.g. a TC message), packets are transmitted only between neighbor nodes. Messages, intended to be forwarded, are (if the node is MPR for the last-hop of the message) re-encapsulated in each node and retransmitted. I.e. the unit of information subject to

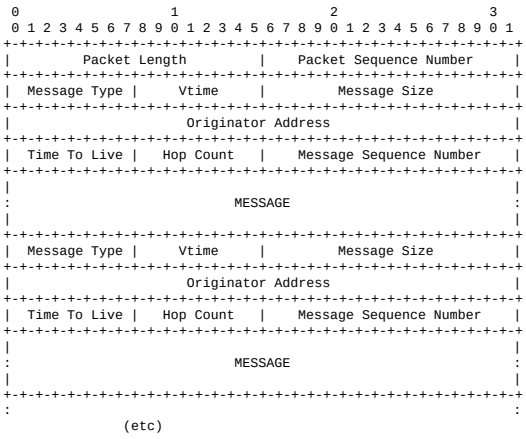

Fig. 2. Generic OLSR packet format. Each packet encapsulates several control messages into one transmission

being forwarded is "messages". The common packet format allows that individual messages be piggybacked and transmitted together in one emission (MTU-size allowing). I.e. TC and HELLO messages may be emitted together, however are processed and forwarded differently in each node (HELLO messages are not forwarded while TC messages are).

It is important to notice, that an individual OLSR control message can be identified by its Originator Address and Message Sequence Number - both from the message header. Hence, disregarding issues of wraparound of the Message Sequence Number, it is possible to uniquely refer to a specific control message in the network. This will become of importance when discussing message signatures in section IV.

Further details on the OLSR packet and message formats can be found in [1].

III. Vulnerabilities

In this section, we will discuss various security vulnerabilities in proactive routing protocol for ad-hoc networks. We will specifically enumerate vulnerabilities in OLSR, however we point out that this section does not emphasize "flaws" in the OLSR protocol. Rather, the vulnerabilities are instances of what all proactive routing protocols are subject to.

One vulnerability, common for all routing protocols operating a wireless ad-hoc network, is that of "jamming" - i.e. that a node generates massive amounts of interfering radio transmissions, which will prevent legitimate traffic (e.g. control traffic for the routing protocol as well as data traffic) on part of a network. This vulnerability cannot be dealt with at the routing protocol level (if at all), leaving the network without the ability to maintain connectivity. Jamming is somewhat similar to that of network overload: a sufficiently significant amount of routing protocol control traffic is lost, preventing that routes can be constructed in the network. In this paper, we will not consider a networks resistance against neither jamming nor traffic overloading.

Our assumption is thus, that the routing protocol is able to consistently provide a correct view of the network topology in each network node. This assumption implies that all nodes in the network correctly implement the routing protocol - specifically that each node correctly processes and emits control traffic.

Thus an attack on the ability to provide connectivity in the network must result from incorrect behavior of, at least, one node in the network. In this context, incorrect means that the node does not process and emit control traffic in accordance with the routing protocol specifications. We note, that in most cases such non-conforming behavior of a node will be due to malice - i.e. specially targeted to interfere with the network connectivity. The node responsible for this incorrect behavior may be either an *intruder* (i.e. a node which is not supposed to be in the network) or a *compromised node* (i.e. a node, which is supposed to be in the network, but which has been modified to be non-conforming with the routing protocol). We also note, that non-conforming behavior of a node may be without malice - e.g. due to a simple malfunction of a node.

When an ad-hoc network is operating under a proactive routing protocol, each node has two different (but related) responsibilities. Firstly, each node must correctly *generate* routing protocol control traffic, conforming to the protocol specification. Secondly, each node is responsible for *forwarding* routing protocol control traffic on behalf of other nodes in the network. Thus incorrect behavior of a node can result from either a node generating incorrect control messages or from incorrect relaying of control traffic from other nodes. This is illustrated in figure 3.

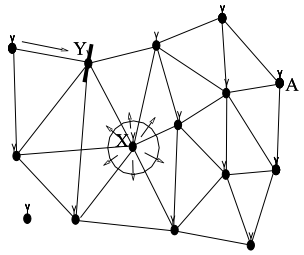

Fig. 3. Two kinds of attacks in a proactive routing protocol: node X generates incorrect information (e.g. advertises node A as a neighbor, while node Y does not relay control traffic for other nodes.

In the remainder of this section, we will investigate how these incorrect behaviors appear in OLSR. We note, that while we employ OLSR for the purpose of our investigations, much of the following applies equally for other proactive routing protocols.

A. Incorrect Control Traffic Generation

OLSR employs, basically, two different kinds of control traffic: HELLO messages and TC messages. In this section, we will describe how a non-conforming node may affect the network connectivity through incorrect generation of HELLO and TC messages.

In general, we observe that with respect to control traffic generation, a node may misbehave in two different ways: through generating control traffic "pretending" to be another node (i.e. *Identity Spoofing*) or through advertising incorrect information (links) in the control messages (i.e. *Link Spoofing*).

B. Incorrect HELLO messages

Identity Spoofing: A node may send HELLO messages, pretending to have the identity of another node. E.g. node X sends HELLO messages, with the originator address set to that of node A, as illustrated in figure 4. This may result in the network containing conflicting routes to node A. Specifically, node X will choose MPRs from among its neighbors, signaling this selection pretending to have the identity of node A. The MPRs will, subsequently, advertise that they can provide "last hop" to node A in their TC messages. Conflicting routes to node A, with possible loops, may result from this.

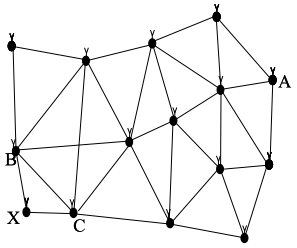

Fig. 4. *Identity Spoofing* of HELLO messages: node X assumes the identity of node A for sending HELLO messages. Nodes B and C may, subsequently, announce reachability to node A through their TC messages.

Link Spoofing: A node may send HELLO messages, signaling an incorrect set of neighbors. This may take either of two forms: if the set is incomplete, i.e. a node "ignores" some neighbors, the network may be without connectivity to these "ignored" neighbors.

Alternatively, a compromized node advertising a neighbor-relationship to non-present nodes may cause inaccurate MPR selection with the result that some nodes may not be reachable in the network.

C. Incorrect TC Messages

Identity Spoofing: A node may send TC messages, pretending to have the identity of another node. Effectively, this implies *link spoofing* since a node assuming the identity of another node effectively advertises incorrect links to the network.

Link Spoofing: A node may send TC messages, advertising an incorrect set of links. This may take either of two forms: if the set is incomplete, i.e. a node "ignores" links to some nodes in its MPR selector set, the network may be without connectivity to these "ignored" neighbors - as well as to neighbors which are reachable only through the "ignored" neighbors. A node may also include non-existing links (i.e. links to non-neighbor nodes) in a TC message. This is illustrated in figure 5.

Link spoofing in TC messages may yield routing loops and conflicting routes in the network.

D. Incorrect Control Traffic Relaying

If TC messages (or routing protocol control messages in general) are not properly relayed, connectivity loss may result. In networks where no redundancy exists (e.g. in a "strip" network), connectivity loss will surely result, while other topologies may provide redundant connectivity. Similarly if a node does not forward data packets (e.g. if intra-node forwarding is impaired), loss of connectivity may result.

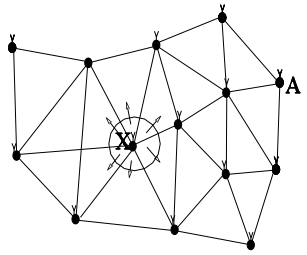

Fig. 5. Node X generates incorrect TC messages, e.g. advertising a link between node X and node A.

IV. Securing OLSR

In this section, we will present a framework, allowing OLSR to resist the security issue which were identified in section III.

We assume that the integrity of the nodes in the network is intact, i.e. that a *trusted* node is not compromized (i.e. that a trusted node is behaving correctly). Thus, the aim is to ensure that only control traffic from trusted nodes is considered in the network, and that the integrity of such control traffic is preserved.

In this section we will first outline the requirements to a cryptographic system, followed by the mechanisms of signing OLSR control messages. This mechanism is cryptography-agnostic, i.e. both asymmetric and symmetric algorithms may be employed as desired. The description furthermore assumes, that the necessary keys to perform signing and verification operations are available to each node.

A. Cryptographic Requirements

The security architecture proposed is mostly cryptography agnostic. I.e. few constraints are enforced on the cryptographic system employed to secure OLSR as described in this paper. In fact, any cryptographic system, satisfying the following two requirements, may be employed:

- a *signature* for a message can be generated in a node using a function:
  `sign(nodeid, key, message)`
- a *signature* for a message can be verified in a node using a function:
  `verif(originatorid, key, message, signature)`

Public-key as well as symmetric shared-secret key systems can be employed. The properties of various cryptographic systems are beyond the scope of this paper.

B. OLSR Signatures

To prevent malicious nodes from injecting incorrect information into the network, a signature is generated by the originator of each OLSR control message and transmitted with the control message. In addition, a timestamp is associated with each signature, in order to estimate a message freshness. Timestamps are discussed in section V.

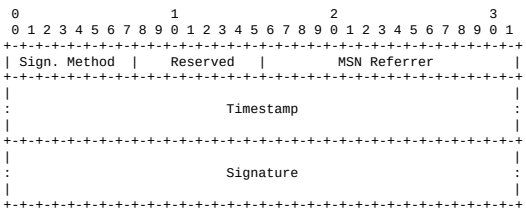

Fig. 6. OLSR signature message format

Thus, upon receiving the control message, a node can determine if the message originates from a trusted node, and if the message integrity is preserved.

Signatures and timestamps are, inheritly, separate entities from OLSR control traffic: while OLSR control traffic serves to acquire and distribute topological information, signatures serve to validate information origins and integrity. Thus, we introduce signatures as a separate type of OLSR control messages, encapsulated and transmitted as described in section II-B.

Signatures are used by a receiving node to authenticate the corresponding OLSR control message: every control message without a matching corresponding signature is rejected. Depending on the properties of the signature method, different levels of authentication and resilience to attacks can be provided. For instance, the highest level of authentication may be provided by using individual asymmetric keys, as the messages advertised as generated from every non-compromised node are uniquely accepted when they indeed originate from this node. Weaker (but less complex or less computationally intensive) systems can be imagined, e.g. employing a shared secret-key system among trusted nodes.

In more details, for each TC or HELLO message generated, a corresponding signature message is generated and transmitted. The format of a SIGNATURE_MESSAGE is specified in figure 6.

To compute a signature corresponding to a TC or HELLO message, the following approach is used. Notice, that this has some similarities with checksum computation:

- the node creates the OLSR control message (HELLO or TC),
- the "current timestamp" is obtained and updated. Timestamps are used for replay protection, and are discussed into detail in section V,
- the signature is computed on the sequence of bytes made up from (i) the TC or HELLO message and (ii) the timestamp. Notice, that for the computation of the signature, the TTL and Hop-Count fields of the TC or HELLO message are considered as set to 0 (zero) since these fields are modified while the message is in transit and, thus, would otherwise interfere with with verification of the message by the receiving node. Thus, the signature is:

```
signature = sign(nodeid,key,
                 < OLSR control message,
                   timestamp > )
```

Upon receiving a matching message and signature pair, the receiving node verifies the signature thus (again, considering the TTL and Hop-Count of the message to be set to zero):

```
verif(originator address, key,
      < OLSR control message, timestamp >,
      signature)}
```

If the verification returns true, then the node proceed to perform timestamp verification, as described in section V.

The signature and the timestamp are contained in an OLSR control message, as illustrated in figure 6 and are transmitted as the data-portion of the general packet format described in section II-B, with the "Message Type" set to SIGNATURE_MESSAGE , the TTL and Vtime fields set to the values of the TTL and Vtime fields of the message to which the signature is associated.

In order to identify correspondence between a TC or HELLO message, the SIGNATURE_MESSAGE contains in the MSN Referrer field the value of the Message Sequence Number of the control message to which this signature is associated. However as pointed out in section II-B, the correspondence achieved by the Message Sequence Number is only unique if possible wraparound of the 16 bit field are disregarded. This is however not a problem since a node further can use the signature verification to check the correspondence between the control message and the signature message:

- Upon receiving a HELLO or TC message, the node holds the message (for an implementation dependent duration), waiting for corresponding signature message.
- Upon receiving a signature message, every message held in the previous step, with the same MSN and originator address as the MSN Referrer and originator address in the signature message, is checked for a signature match. If a signature match is found, the timestamp is further verified, as described in next section. If both signature and timestamp are validated, the message is accepted and processed following the rules of the OLSR protocol. If not, both the signature message and the control message(s) are held.

The *Sign. Method* fields specifies which method, among a predefined set, is being used to generate the summary of the control message, as well as the actual signature. This includes informations about the keys to use, the hashing function used for signature, and timestamp methods.

V. TIMESTAMPS

A common problem in distributed systems with non-secure communication channels is that, even assuming a signature is checked (thus ensuring authentication of the source of a message), *replay* of previously transmitted messages is possible by an intruder. For instance, an intruder in a network may replay one day worth of control messages, which, if they cannot be identified as "old" (for some definition of old), will be accepted as valid because they are properly signed. This may easily disrupt the protocol functioning and thus the network integrity.

Timestamps or *nonces* are a commonly used means to prevent replay attacks, (e.g. Kerberos [15] or [2] for more examples from a library of authentication protocols), and are indeed necessary (e.g. [5]). The idea is to device a *proof of freshness*, such that older pieces of information can be detected and rejected. In OLSR, MSSN and ANSN are already used for achieving those goals in the context of allowing the routing protocol to determine which information is more recent. However while these sequence numbers are sufficient for the basic routing protocol functioning, they are not sufficient to provide full security: each

are a 16 bit field, which implies that wrap-around happens too frequently to provide efficient protection against malice from intruding nodes.

In this section, we describe several timestamp algorithms, providing different levels of security at the expense of different costs. For the purpose of our discussion, we employ the following terminology:

- a *clock* is the "device" (hardware, software) within a node keeping track of the time,
- a *timestamp* is the value of a clock, recorded in a piece of information (e.g. a message), at the time of generation of the information.

Further discussion of timestamp methods can be found in [6]. Commonly, the following methods are employed for providing timestamps:

- *real time:* a clock expresses time in some natural resolution such as seconds, microsecond, etc.
- *logical time:*, i.e. a clock is incremented each time an event occurs, such as message generation.

Note that these are just different ways to express the same idea of time: the basic property being that time is monotonically increasing, and that a upon receiving a message containing a timestamp, the receiving node has some idea of how the value in the timestamp compares to the value of the clock - i.e. what the timestamp should be.

For each message being emitted by a node, an unique timestamp, T_{sender}, is included. Let T_e denote the value of the clock in a receiving node, around which the timestamp, T_{sender}, in a received message is expected to be. Then, a more formal expression of a message being "not too old" may be:

$$|T_{sender} - T_e| < \delta_{max} \tag{1}$$

where δ_{max} is a constant used to limit the timestamp discrepancy while allowing for some small deviation. Thus, (1) provides a simple framework for *replay check*, i.e. checking if a received message is original, or is a replay of a previous message.

The replay check in (1) can be complemented, in order to also prevent replays within a small time-scale (i.e. replays within a delay less than δ_{max}), by maintaining a *signature table*. The signature table contains the signatures of the most recently received messages, for a duration greater or equal to δ_{max}. If the signature of a received message is already in the *signature table*, it is ignored since the message has already been received and processed. This is similar to the duplicate table of OLSR (ensuring that TC messages are processed and forwarded once). Indeed, the functionalities of both the signature table and the OLSR duplicate table could be merged.

The way in which timestamps are generated is not necessarily obvious, as they assume either synchronous real-time timestamps, non-volatile timestamp, or they implicitly require a *challenge-response* protocol.

In the following, we present different methods for generating timestamps. Two of these are derived from ideas described in [6], the last being a derivative of the well-known Needham-Schroeder public key authentication algorithm [11]. The different methods introduces different levels of complexity, cost and security tradeoffs.

1) No timestamps: If no replay protection is desired, nodes may just set the timestamp to be 0 when generating messages, and not check the timestamps upon receiving.

2) Real-time timestamps: A conceptually simple way to generate timestamps (although not the easiest to implement), is to use a real time clock in each node, assuming some kind of synchronization. This can be achieved in several ways:

- by having a "safe" source of time in each node, sufficiently precise and with sufficiently little drift. This could e.g. be in form of an atomic clock, access to the time as obtained by a GPS device - or simply a normal clock in each node, adjusted by means of wristwatch.
- by clock synchronization.

The criterion for accepting a message is indeed simply

$$|T_{sender} - T| < \delta_{max} \tag{2}$$

where T is the value of the clock on the receiving node.

With respect to clock synchronization, a dilemma arises: as [6] notes, timestamps are used for authentication, but secure clock synchronization itself requires authentication. This is overcome by performing authentication and clock synchronization operations at the same time. However the main problem with secure clock synchronization in a MANET is, that many algorithms require a fixed fraction (percentage) of non-compromised nodes in order to operate. Since in a MANET, an intruding node can impersonate as many fictional nodes as it wishes (under the limitation that those fictional nodes have keys known to the network), a guaranteed fraction of non-compromized nodes is unobtainable. Even a clock synchronization algorithm, such as [5], which doesn't require any such fraction of correct nodes to *run* properly, provenly cannot bound the necessary delay of synchronization when a new node wishes to *join* (e.g. participate in the network for the first time). This is quite problematic in a wireless ad-hoc network, since nodes are expected to be able to leave and join at any time.

Since secure time synchronization is not simpler than the protocol described in section V-.4, this approach will not be pursued further in this paper.

3) Non-volatile timestamp: A way to provide weak timestamps is, to have the clock of each node of the network maintained in non-volatile memory, initialized the first time a node signature keys is used after generation.

The value of this clock is, then, used as timestamp in each message signed, after which the value is incremented. While the sender maintains the clock in non-volatile memory, receivers maintain a table containing the maximal timestamps received from all nodes in the network.

In the receiver node R, the algorithm for processing a message from sender S with timestamp T_S, is the following:

- R keeps the value of T_S^R, the highest timestamp from S it ever received, in non-volatile memory.
- If R hasn't received any value from S, R considers the highest timestamp received from S to be $T_S^R \leftarrow 0$.
- If $T_S > T_S^R - D$, the message is accepted, otherwise it is rejected and not processed further. D is some fixed (small) constant, to allow for out of order receiving of message. D must be tuned accordingly to the specifics of the ad-hoc network.

- T_S^R is updated: $T_{S,new}^R \leftarrow max(T_S, T_S^R)$

These are security-wise weaker timestamps since, if communication between the sender and the receiver is broken for some time, all the messages from the sender can be replayed to the receiver. This is especially true if the sender and the receiver are in different, non-connected, networks. The advantage is, that as soon as the receiver and the sender are able to communicate with each other, only limited replay is possible. This replay can further be suppressed with the signature table, described previously.

In OLSR, non-neighbor nodes may never exchange messages (nodes, which are not selected as MPR by any other nodes exchange messages only with their neighbor nodes). In this case, the timestamps would never be updated. Thus it would be necessary that each node periodically broadcasts at least an empty message in order to provide synchronization.

Note that a variant using a local "wall clock" time instead of incremented timestamps is possible, and could allow more stringent checks, although the algorithm still remains vulnerable.

4) Timestamp Exchange Protocol: This part describes a timestamp exchange protocol, which essentially mixes a distributed *challenge-response* protocol with timestamps. It is a variation over the Needham-Schroeder public key protocol [11], albeit with a superset of the information (and includes, for instance, the necessary correction of the protocol proposed in [10]), using signatures instead of encryption, and using timestamps instead of nonces.

The assumptions for the protocol are the following:

- Each node, A, keeps a clock, T_A, whose value is used in the timestamp fields of generated messages, represented by a identical fixed amount of bytes for all nodes. The clock increases monotonically with each message sent (and with wall clock). At a given wall clock time t, the clock in the node A is denoted $T_A(t)$.
  The clock, T_A, is also used as a *nonce* and thus should be initialized (fully, or in part) with random values.

First a simplified version of the protocol is given, illustrating the only key ideas of the protocol, limited to two nodes A, B. We use the notation $X \rightarrow Y : \{Z\}_{S_X}$ meaning "X sends to Y the message Z signed with the private key of X":

1) At t_0: $A \rightarrow B : \{A, T_A(t_0)\}_{S_A}$
2) At $t_1 > t_0$, B has already received the previous message and sends: $B \rightarrow A : \{B, T_B(t_1), A, T_A(t_0)\}_{S_B}$
3) At $t_2 > t_1$, A has received the previous message and sends: $A \rightarrow B : \{A, T_A(t_2), B, T_B(t_1)\}_{S_A}$.

The idea is that at t_2 (step 3), A had received the message sent in step 2, and thus observed that a recent version of its timestamp $T_A(t_0)$ was included in a message, authenticated to be from B. Thus, A can safely assume $T_B(t_1)$ is a recent value of the timestamp from B, posterior to t_0. This relies on A properly generating initial values of clock T_A, i.e. not (or with low probability) repeating values over the time.

Likewise, upon receiving the message sent from A to B in step 3, B concludes, like A, that it has now a recent authenticated value of T_A. After those steps are complete, A and B both have knowledge about relatively recent values of each others respective timestamps, which are not the result of replays (or with very low probability). In that case, we say that the *handshake* is completed.

A detailed parallel version of the algorithm now is given. It is "parallel" in the sense that the same message, sent by a node A to perform the previously illustrated handshake, this time is sent to several nodes (ideally all) in the network, rather than to an individual node B. Also some provisions are taken, for being able to practically perform timestamp check, and for switching to new timestamp intervals.

The protocol relies on one unique new kind of message, a *timestamp exchange message*, being flooded periodically by each node. When maximal security is desired, the message should be a transmitted by pure flooding to the network.

It is assumed that each node keeps a table of the informations from the latest timestamp exchange message it received from each node. This table is called the *timestamp table*. Thus considering a node A, for each node B, node A records the following information:

- a boolean H_B^A indicating whether the handshake with B has been completed,
- the timestamp, T_B^A, from the latest *timestamp exchange message* received by node A from node B (in case the handshake is completed), or the list of the latest timestamps $T_{B,j}^{*A}$ received (in case the handshake is not completed),
- a set of different timestamp interval tuples for i=1,..,n_B^A:

$$(T_{B,min,i}^A, T_{B,max,i}^A, E_{B,i}^A)$$

  where $E_{B_i}^A$ is an expiration time, indicating the tuple should only be used until the time reaches $E_{B_i}^A$ (when the handshake is completed).

In node A, each timestamp interval tuple of B, describes a valid interval for timestamps of B. There are several such timestamp interval tuples for B (several i), in order to allow for timestamp interval changes. Such a change would occur, when for instance the node would decide to regenerate a clock from scratch. The timestamp interval tuples are used with the following *timestamp check* "algorithm":

In node A, at t, a timestamp $\mathcal{T}_B$ from a message from B is valid if and only if:

- there exists one i, such that
- $(T_{B,min,i}^A \leq \mathcal{T}_B \leq T_{B,max,i}^A$ and $t < E_{B,i}^A)$

This *timestamp check* does not apply to timestamp exchange messages themselves, described below.

At node A, given two valid timestamps from B, $\mathcal{T}_B$ and $\mathcal{T'}_B$, an ordering relation can be established for comparison:

- Let j and j' be the indexes such that: $(T_{B,min,j}^A \leq \mathcal{T}_B \leq T_{B,max,j}^A)$ and $(T_{B,min,j'}^A \leq \mathcal{T'}_B \leq T_{B,max,j'}^A)$
- Then it is decided that $\mathcal{T}_B > \mathcal{T'}_B$, if and only if:
  - $j > j'$
  - or: $j = j'$ and $\mathcal{T}_B > \mathcal{T'}_B$

This is used for determining which of two messages, to which the timestamps relate, is the most recent.

For protocol completeness, in node A, the timestamp T_B^A is said to be *orphaned* when H_B^A is $false$ or when T_B^A does not pass the timestamp check with any of the timestamp intervals $(T_{B,min,i}^A, T_{B,max,i}^A, E_{B,i}^A)$. This can occur when some (or all)

of those intervals expire, usually meaning that communication between node A and node B is broken.

This yields a formal definition of "completion of handshake between A and B": a handshake is complete for node A when B has a timestamp T_B^A which is not orphaned. Each time H_B^A was true and the timestamp T_B^A becomes orphaned because of timestamp interval, H_B^A is updated as: $H_B^A \leftarrow false$.

The protocol relies on two parts: the generation and the processing of timestamp exchange messages. Each of these parts will be described in the following.

The *generation algorithm* of the timestamp exchange messages is as follows: node A sends periodically one *timestamp exchange message*, containing:

- its current clock T_A^{new}. T_A^{new} must not be orphaned with respect to the bounds set out in the following item,
- its current timestamp bounds set: n_A^{new}, and for $i = 1 \ldots n_A^{new}$,

$$(T_{A,min,i}^{new}, T_{A,max,i}^{new}, D_{A,i}^{new})$$

  where $D_{A,i}^{new}$ is the maximal duration for which the tuple should be kept,
- the content of its *timestamp table*, tuples: (X, T_X^A) for each node X from which it has received a timestamp. Note that for each X with which the handshake is not completed, there might be several $(X, T_{X,j}^{*A})$. In this case, once the $(X, T_{X,j}^{*A})$ have been sent, the tuples are removed.
- (the message is also signed)

Node A also records the latest timestamp bounds set it sent:

$$(T_{A,min,i}, T_{A,max,i})$$

The *processing algorithm* of the timestamp exchange messages for a node A, receiving a message from node B, is as follows:

- check, for the entry of B (if it exists) in the timestamp table to determine if H_B^A was $true$ but the T_B^A has become orphaned. If T_B^A has become orphaned, then $H_B^A \leftarrow false$
- If no reported timestamp from A, inside the timestamp exchange message pass the timestamp check in A (or if there is no T_A in the message):
  - If no entry for B was recorded or if H_B^A is $false$, the timestamp from the message T_B, is added to the list of the timestamps $T_{B,j}^{*A}$.

  The idea here is, that the node, B, hasn't provided enough proof of freshness, for A to accept the timestamp intervals. However T_B should be kept such that in next message from A it would serve as proof of freshness. All T_B received should be kept since some could constitute invalid replays.
- Otherwise: The handshake is certain to be completed and the timestamp bounds are updated if necessary:
  - if no value T_B^A was recorded or if H_B^A was $false$, or if the new timestamp T_B of the message is greater than the latest timestamp recorded T_B^A (with the timestamp comparison rules given previously):
    - T_B^A is updated with the timestamp from the message: the previous list $T_{B,j}^{*A}$ is emptied and: $T_B^A \leftarrow T_B$.
    - The timestamp bounds for B are updated with the values from the timestamp exchange message. $n_B^A \leftarrow n_B$, and for $i = 1 \ldots n_B$: $(T_{B,min,i}^A, T_{B,max,i}^A, E_{B,i}) \leftarrow (T_{B,min,i}, T_{B,max,i}, D_{B,i} + t)$ where t is the wall clock at node A.
  - $H_B^A \leftarrow true$

It is expected that the "timestamp bound set" of a node is limited to one interval $(T_{A,min}^1, T_{A,max}^1)$, and only occasionally updated when a new timestamp interval $(T_{A,min}^2, T_{A,max}^2)$ is generated. The transition is typically the following:

- A advertises the interval $(T_{A,min}^1, T_{A,max}^1)$ and generates timestamps in this interval.
- for some duration, A advertises interval $(T_{A,min}^1, T_{A,max}^1)$ and a new interval $(T_{A,min}^2, T_{A,max}^2)$. A will still generate timestamps from the first interval, to waiting for the new interval to be updated in receiving nodes.
- for some small duration, A advertises both interval $(T_{A,min}^1, T_{A,max}^1)$ and interval $(T_{A,min}^2, T_{A,max}^2)$. A now generates timestamps from the second interval, while it keeps advertising the old interval.
- A advertises only interval $(T_{A,min}^2, T_{A,max}^2)$

Note that variations are possible, such as also updating timestamp table on receiving messages such as HELLOs, introducing some maximum deviation δ_{max} from the last received timestamp, or using local wall clock (possibly with a random offset), instead of an incremental counter. Optimizations are possible, to avoid sending lists of timestamps of the same node before handshake completion (such as sending immediately an time exchange message), along some with denial of service detection with respect to the handshake protocol.

VI. Public Key Acquisition

The security mechanism described in section IV assumed that provisions existed such that any node in the network would be able to acquire the keys required for verification of signatures of any other node. In the case where shared-secret systems are employed, key-distribution must be performed such that the shared secret key cannot be acquired while in transit (nor, preferably, recovered from a node after distribution).

Public-key systems have a strength in that public keys can be exposed to anyone - the difficult part being for a node to validate that a public-key, claiming to belong to a specific node, in fact does belong to that node. A key problem in public key cryptography is thus the distribution of keys in a way such that the public keys can be trusted - i.e. that they belong to the node to which they claim that they belong. In the context of this paper, this translates into the ability for a node to trust received information.

In this section, we will outline two simple public key infrastructure (PKI) systems for MANETs. They both serve the purpose of making public keys available to nodes in the network in a way such that the authenticity of the keys can be trusted. The two PKIs differ mainly in that the first is "proactive", i.e. it proactive aims at diffusing public-key information to nodes in the network, while the second is "reactive: nodes request keys when required only.

We note, that the PKI systems outlined here are "basic" in the sense that they concern them self with simple key distribution.

A. A Simple Proactive PKI for OLSR

For the purpose of distributing and trusting public keys, we propose a simple proactive PKI for OLSR.

This PKI operates with three classes of nodes, as seen from an individual nodes point of view:

Untrusted Nodes
: A node, *a*, considers another node, *x*, as an "untrusted node" if the public key of *x* is not known by node *a* or if the public key of node *x* is known, but not validated by a signing authority in the network. I.e. signature messages, received from an untrusted node, cannot be verified. Notice, that at network initialization, all nodes except the signing authority and the node itself will be "untrusted", from the individual nodes point of view

Trusted Nodes
: A node, *a*, considers another node, *x*, as a "trusted node" if the public key of *x* is known by node a and the public key of node *x* is validated by a signing authority in the network. I.e. signature messages, received from trusted nodes, can be verified.

Signing Authorities
: A "signing authority" is a node, which has the special property that its public key is a priori known by all nodes in the network. A signing authority has special responsibilities for the network, namely to:
  - allow new nodes to register their public keys in a secure fashion (typically through manual authentication), whereby a new node becomes a "trusted node",
  - periodically distribute certificates, containing:
    - a list of public keys for all "trusted nodes",
    - a signature, verifying that the message, containing the public keys of the "trusted nodes" was, indeed, issued by the signing authority and was not modified while in transit.

Each node that wishes to participate in the network is required to register its public key with a signing authority. The signing authority will issue certificates periodically, which will be broadcast to the entire network. Nodes, receiving the certificates, will store these for a specified amount of time, after which they expire. Hence, periodic refresh of certificates is required.

Upon network initialization, no nodes know any public keys other than that of the signing authority (and, of course, their own). Thus, disregarding control traffic from the signing authority, all nodes will by default ignore control traffic from each other and. Thus, no nodes will select MPRs - and no broadcast messages will be forwarded. Control traffic from the signing authority will be accepted by its neighbors since they know the public key of the signing authority in advance. Thus, until the signing authority starts broadcasting certificate messages, no network formation will take place. Unless special provisions are made, only neighbor nodes of the signing authority will ever receive the broadcast certificates: since successful verification of a signature is a criteria for accepting any control messages, 2-hop neighbors of the signing authority will never accept control messages from 1-hop neighbors of the signing authorities. This implies that a symmetric link between 1-hop and 2-hop neighbors of the signing authority will never be established. The signing authority will therefore never select MPRs and, thus, that certificates will never be broadcast into the network.

Thus, to enable network initialization, special provisions for accepting some control messages without validation of signatures must be provided. The desired goal is to allow for MPR flooding to take into account the fact that broadcast messages should be able to reach also untrusted nodes in the network. Thus, to enable this, the following additional conditions apply

- A node must accept unsigned HELLOs from untrusted neighbors (i.e. neighbors, whos public key is not yet known to a node). Such HELLO messages are accepted under the restriction that:
  - asymmetric and symmetric links are considered as asymmetric and symmetric, respectively,
  - MPR links are considered as symmetric only (i.e. do not affect the MPR selector set),
  - lost links are ignored
- A node must maintain a "trusted neighborhood" containing information about links to the trusted nodes in its neighborhood
- A node must maintain an "untrusted neighborhood", containing information about links to the untrusted nodes in its neighborhood.
- A node must, from among the trusted neighborhood, perform MPR selection as specified.
- A node must, periodically, transmit HELLO messages, including the trusted neighbors (with status: asym, sym and MPR as appropriate) and untrusted neighbors (with status: asym, sym only)

A nodes 2-hop neighborhood will contain both "trusted" and "untrusted" nodes. MPRs are selected from among the trusted nodes such that - if possible - all nodes in the 2-hop neighborhood are covered.

Thus, upon network initialization, the signing authority will transmit its certificate, which is received by its 1-hop neighbors. Following HELLO message exchange, the 1-hop neighbors will accept the untrusted 2-hop neighbors as symmetric (but not select MPRs among them). The signing authority will then select MPRs from among the 1-hop neighborhood such that the next broadcast certificate will reach the 2-hop neighbors etc. The certificates will thus, upon network initialization, propagate from the signing authorities and towards the edges of the network.

Notice, that a information coming from "untrusted" nodes is only used to handle "untrusted" nodes: MPR selection etc. is performed only among "trusted" nodes, as is MPR selector information only diffused about "trusted" nodes.

Also notice, that no explicit mechanisms for revoking keys is presented. To facilitate key revocation, certificate messages may be equipped with a sequence number, associated with the set of keys advertised. Whenever the set changes (keys are added or removed) the sequence number is incremented,

and included in following certificate messages. Upon receiving a certificate messages, nodes can distinguish between older and newer information, and remove expired keys. In order to counter possible replay attacks, timestamps - as described in previous sections - can be employed.

B. A Simple Reactive PKI for OLSR

Unlike the previous PKI, designed to be a proactive distribution of public keys, this section describes a simple reactive PKI for OLSR. The behavior is reactive, that is, when a node does not possess some required information itself, it diffuses a query to the network. In this case, the required information is the public keys of some nodes in the network.

The same framework as in previous section is assumed, and the same notation as in the section V is used.

Two new types of messages are introduced, *Key request* and *Key reply*.

- A *key request* is a message from a node A, with a nonce N_A initialized with random values for each request, a list of nodes B_i for which the public key is need: $A \rightarrow all : \{A, N_A, B_1, ..., B_n\}_{S_A}$
- Upon receiving such a request message, a signing authority:
  - first checks the signature of the message A, if it has the public key of S_A.
  - if the public key of at least one B_i in request is known by the authority, a *Key reply* is generated. The reply includes all the public keys it knows: $P \rightarrow all : \{P, A, N_A, (B_{i_1}, PK_{B_{i_1}}), \ldots (B_{i_m}, PK_{B_{i_m}})\}_{S_P}$
- Upon receiving such a reply message, the originating node A, performs the following checks:
  - that the destination of the message is indeed A.
  - that P is a signing authority that it trusts.
  - that the signature of the message with S_P is correct.
  - that the nonce N_A was a nonce it recently used. This is to avoid replay.

  If those checks succeed, node A finally updates its public key database with the newly acquired keys.
- In order to ensure proper delivery of *Key request* and *Key reply* messages, pure flooding is used. I.e. a node will retransmit a received message the first time it receives it.

As with the proactive PKI, considerations regarding key revocation are not presented. These features, however, can be fashioned through lifetime of the public keys and periodic refreshing through renewed request-reply cycles.

VII. Conclusion and further work

In this paper, we have examined the issues related to security of a proactive link-state protocol such as OLSR. Some of the insights provided are general to a larger class of protocols (link-state protocols, or proactive protocols), while others are related directly to optimizations specific to OLSR (such as to MPR flooding). The source of vulnerability of OLSR, which is common to link state protocols in general, was identified: introduction of incorrect topology information (either locally or globally). To secure the protocol against foreign nodes with malicious intent, a framework was described using authentication checks. This framework included a way to diffuse authentication of OLSR protocol messages, a discussion and description of algorithms for timestamps to prevent the difficult problem of replay attacks, in which a malicious node "replays" previously valid traffic in the network. Finally, the framework included a description of two algorithms for public keys acquisition. One of these is proactive, fitted to the link state behavior in general and to OLSR in the specifics, illustrating the other difficult problem of making the PKI interact with the routing protocol during "bootstrap". The other algorithm for key acquisition is a simple, reactive, but potentially expensive (in terms of overhead), protocol.

References

[1] Cdric Adjih, Thomas Clausen, Philippe Jacquet, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum, and Laurent Viennot. Optimized link-state routing protocol. draft-ietf-manet-olsr-08.txt, March 3 2003, Work in progress.

[2] John A. Clark and Jeremy L. Jacob. A survey of authentication protocol literature. Technical Report 1.0, 1997.

[3] Thomas Clausen, Gitte Hansen, Lars Christensen, and Gerd Behrmann. The optimized link state routing protocol - evaluation through experiments and simulation. In *Proceeding of Wireless Personal Multimedia Communications*. MindPass Center for Distributed Systems, Aalborg University, Fourth International Symposium on Wireless Personal Multimedia Communications, September 2001.

[4] Thomas Clausen, Philippe Jacquet, and Laurent Viennot. Investigating the impact of parital topology in proactive manet routing protocols. In *Proceeding of Wireless Personal Multimedia Communications*. MindPass Center for Distributed Systems, Aalborg University and Project Hipercom, INRIA Rocquencourt, Fifth International Symposium on Wireless Personal Multimedia Communications, November 2002.

[5] Danny Dolev, Joseph Y. Halpern, Barbara Simons, and Ray Strong. Dynamic fault-tolerant clock synchronization. *Journal of the ACM*, 42(1):143–185, 1995.

[6] L. Gong. Variations on the themes of message freshness and replay. In *Proceedings of the IEEE Computer Security Foundations Workshop VI, Franconia, New Hampshire*, pages 131–136, June 1993.

[7] Manel Guerrero Zapata and N. Asokan. Securing Ad hoc Routing Protocols. In *Proceedings of the 2002 ACM Workshop on Wireless Security (WiSe 2002)*, pages 1–10, September 2002.

[8] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking*. Rice University, MobiCom 2002, September 2002.

[9] J. G. Jetcheva, D. Johnson, D. Maltz, and Y.C. Hu. Dynamic source routing (DSR). Internet Draft, draft-ietf-manet-dsr-08.txt, February 24 2003, Work in progress.

[10] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.

[11] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[12] R. Ogier, M. Lewis, and F. Templin. Topology dissemination based on reverse-path forwarding (tbrpf). Internet Draft, draft-ietf-manet-dsr-07.txt, March 3 2003, Work in progress.

[13] C. E. Perkins, E. M. Royer, and S. R. Das. Ad hoc on-demand distance vector (AODV) routing. Internet Draft, draft-ietf-manet-aodv-13.txt, February 17 2003, Work in progress.

[14] Amir Qayyum, Laurent Viennot, and Anis Laouiti. Multipoint relaying: An efficient technique for flooding in mobile wireless networks. Technical report, Project HiPERCOM, INRIA Rocquencourt, 2000. INRIA research report RR-3898.

[15] J. Steiner, C. Neuman, and J. Schiller. An authentication service for open network systems, 1988.