LHAP: A Lightweight Hop-by-Hop Authentication Protocol For Ad-Hoc Networks

Sencun Zhu[1], Shouhuai Xu[2], Sanjeev Setia[1]*, and Sushil Jajodia[1]

[1] Center for Secure Information Systems, George Mason University, Fairfax, VA 22030.
{szhu1,setia,jajodia}@gmu.edu

[2] Department of Information and Computer Science, University of California at Irvine, Irvine, CA 92697.
shxu@ics.uci.edu

Abstract. Most ad hoc networks do not implement any *network access control*, leaving these networks vulnerable to *resource consumption* attacks where a malicious node injects packets into the network with the goal of depleting the resources of the nodes relaying the packets. To thwart or prevent such attacks, it is necessary to employ authentication mechanisms that ensure that only authorized nodes can inject traffic into the network. In this paper, we present LHAP, a scalable and light-weight authentication protocol for ad hoc networks. LHAP is based on two techniques: (i) hop-by-hop authentication for verifying the authenticity of all the packets transmitted in the network and (ii) one-way key chain and TESLA for packet authentication and for reducing the overhead for establishing trust among nodes. We analyze the security properties of LHAP. Furthermore, our performance analysis shows that LHAP is a very lightweight security protocol.

1 Introduction

In ad hoc wireless networks, no base stations exist and each mobile node acts as both a router and a host. Nodes in an ad hoc network can communicate with each other at any time, subject to connectivity limitations. Currently, most ad hoc networks do not have any provisions for restricting or regulating the traffic that flows through a node, i.e., they do not implement any *network access control*. This leaves these networks vulnerable to *resource consumption* attacks where a malicious node injects packets into the network with the goal of depleting the resources of the nodes relaying the packets. For example, since mobile hosts are usually battery powered, they are susceptible to battery exhaustion attacks [30].

Clearly, a network access control capability is essential for ad hoc networks in an adversarial environment such as a battlefield. A resource consumption attack can be especially effective if a packet injected into an ad hoc network by a malicious node ends up being multicast or broadcast throughout the network. For example, the operation of most routing protocols involves steps in which a control packet, e.g., a route request packet, is broadcast to all nodes. Moreover, many applications for ad hoc networks are group-oriented and involve collaborative computing; thus multicast communication is likely to increase in importance as multicast routing protocols for ad hoc networks become more mature.

* also with Dept. of Computer Science, George Mason University

Most of the routing protocols that have been proposed for ad hoc networks do not address the issue of network access control. In these protocols, a node trusts that its neighbors will forward packets for it and also assumes that the packets it receives from its neighbors are authentic. This naive trust model allows a malicious node to inject erroneous routing requests or routing updates into a network, which can paralyze the entire network. To deal with such attacks, recently several researchers [5, 9, 10, 32] have proposed security extensions to existing routing protocols that include mechanisms for authenticating the routing control packets in the network.

To the best of our knowledge, however, none of the proposed secure routing protocols include any provisions for authenticating data packets. One reason for this is that data packets are typically unicast, and therefore simply restricting the number of hops a packet can take will limit the effectiveness of a resource consumption attack. However, this argument does not apply to multicast data packets. As such, we believe that it is important to provide network access control for both data and control packets.

To provide full network access control, an intuitive solution is to authenticate all packets so that a node only forwards packets from authorized nodes. In this way, an attacker cannot insert spurious information into the network at will, because its neighbors will drop these packets immediately once they fail to verify their sources. To allow this scheme to work, a simple solution is to use a network-wide key shared by all nodes [10, 2]. Each node uses this shared key to compute message authentication codes (MACs) on the packets it sends. Nodes receiving these packets can use the same key to verify the MACs and hence the authenticity of the packets. Despite its simplicity, this scheme has several disadvantages. First, an outsider breaking the global key could break down the protocol. Second, it is difficult to identify the compromised node when the global key is divulged. Third, it is expensive to recover from a compromise because it will involve a global re-key, i.e., generating a new key and distributing it to all the legitimate nodes securely and reliably in a timely fashion.

Another option is to use authentication techniques based on public key or asymmetric key cryptography. Although such techniques have become quite mature and are widely deployed in wired networks such as the Internet, they usually do not adapt well to ad hoc networks. In wireless ad hoc networks with high node mobility, the neighbor set of a node may keep changing; therefore, the frequency and hence the cost for performing mutual authentication between nodes is much greater than that in wired networks that do not have node mobility. Further, the resources of a mobile node such as battery power, computational capacity and bandwidth are usually quite constrained. All these facts make most of the authentication protocols proposed in literature impractical for use in ad hoc networking. Therefore, the challenge here is to design a lightweight authentication protocol for the more vulnerable and yet more resource-constrained environment of an ad hoc network.

In this paper, we present LHAP, a scalable and efficient authentication protocol for ad hoc networks. To prevent resource consumption attacks, LHAP implements lightweight hop-by-hop authentication, i.e., intermediate nodes authenticate all the packets they receive before forwarding them. Using LHAP, a node joining an ad hoc network only needs to perform some inexpensive authentication operations to bootstrap a trust relationship with its neighbors. It then switches to a very lightweight protocol for subsequent traffic authentications.

LHAP is transparent to and independent of the network routing protocols. It can be thought of as residing in between the data link layer and the network layer, providing a layer of protection that can prevent or thwart many attacks from happening, including attacks on ad hoc routing protocols made possible by the lack of support for packet authentication in these protocols.

The rest of this paper is organized as follows. In Section 2, we provide some background on one way hash chains and TESLA that are necessary for understanding the techniques used in our protocol. We present the details of the LHAP protocol in Section 3, and analyze its security in Section 4. In Section 5, we analyze the performance of our protocol, and show several possible optimizations for its real deployment in Section 6. Finally, we discuss related work in Section 7, and present our conclusions in Section 8.

2 Background

LHAP makes use of one-way key chains [17] for traffic authentication and TESLA [26, 25] for bootstrapping trust. We briefly review these techniques in this section.

One-way Hash Chains Since its first use by Lamport [17] for one-time passwords, one-way key chains have been widely used in cryptography. A one-way key chain is a chain of keys generated through repeatedly applying a one-way hash function on a random number. For instance, if a node wants to generate a key chain of size N, it first randomly chooses a key, say $K(N)$, then computes $K(N-1) = F(K(N))$, $K(N-2) = F(K(N-1))$,..., repeatedly until it gets $K(0) = F(K(1))$. Here F is a pseudo random function [7] that has the property of one-wayness, that is, if $y = F(x)$, it is computationally infeasible to compute x given y and F. So given $K(i)$, anybody can compute $K(i-1)$, $K(i-2)$,...,$K(0)$ independently, but they cannot compute any keys in $K(i+1), K(i+2), ..., K(N)$.

To use a one-way key chain for authentication, a sender first signs the last value in the chain, i.e., $K(0)$ above, with its private key so that anybody who knows its public key can verify the signature and hence the authenticity of $K(0)$. Then the sender discloses keys in the chain in an order reverse to that of its generation. A receiver can authenticate $K(j)$ by verifying $K(j-1) = F(K(j))$ if it has $K(j-1)$. Furthermore, if a receiver did not receive $K(j-1)$ and the last key it authenticated is $K(i)$, where $i < j-1$, it can still authenticate $K(j)$ by verifying $K(i) = F^{j-i}(K(j))$. This property is very useful because it means the authentication scheme can tolerate packet losses.

TESLA TESLA [26, 25] is a broadcast authentication scheme proposed by Perrig et al, and it has received extensive attention recently. The main technique used in TESLA is a one-way key chain along with delayed key disclosure. After bootstrapping an authentic key from a one-way key chain between the sender and its receivers using a digital signature, TESLA uses MACs for subsequent broadcast authentications but with delayed key disclosure. In the basic scheme of TESLA, a sender uses a key K from its key chain as the MAC key to compute a MAC over packet $P(i)$, and then attaches the MAC to $P(i)$. The disclosure of the key K is in the next packet $P(i+1)$, which allows the receivers to verify the authenticity of K and hence the MAC of $P(i)$. If both K and the MAC are correct, and if the packet $P(i)$ is guaranteed to be received before $P(i+1)$ was sent, the receivers then believe $P(i)$ is authentic.

From this description, we can see clearly that the central security issue in TESLA is a receiver's ability to determine the sending time of each packet. This is called security condition in TESLA. TESLA solves this issue through periodical key disclosure and loose time synchronization. In TESLA, time is divided into many equal length intervals. In each time interval the sender discloses one key from its key chain. For example, if the start time is s, the time interval is T, the sender can publish $K(i)$ at time $s + i * T$. The receivers are required to be loosely synchronized with the sender. Here “loosely” means they know the upper bound of the synchronization error between any two nodes. Given this synchronization error and the sender’s MAC key disclosure interval, a receiver can determine whether the sender has already published the MAC key for a packet it just received. If its MAC key is (possibly) already disclosed, the receiver will drop the packet; otherwise, it will use the MAC key in this packet to verify an earlier packet, and then buffer this packet until a later packet containing its MAC key arrives.

One potential shortcoming of the original TESLA protocol is that the receivers cannot authenticate a packet immediately on its arrival. Its variant [25] enables immediate authentication on the receiver side, but it requires packet buffering at the sender side.

3 A Lightweight Hop-by-hop Authentication Protocol (LHAP)

In this section, we first describe the assumptions we made during the design of our protocol. Next we give an overview of the design goals and basic operation of LHAP. Finally, we discuss the operations of LHAP in detail.

3.1 Assumptions

First, we assume the wireless network links are bidirectional, i.e., if node A can hear node B, node B can also hear node A. This is generally true when the nodes use omni-directional antennas and have the similar power levels. Note that this assumption is also necessary for most of the current routing protocols and applications to work in wireless networks.

Second, we assume that the following condition holds: a packet sent by a node is received by a neighboring node before a third node can replay the packet to it, unless the neighbor under consideration has dropped the packet. For example, if node A sends a packet to its neighbor B, then it is assumed impossible that another node P hears the packet, (potentially) modifies it, and re-sends it to node B such that the replayed packet arrives at node B before the original packet does. We assume that if node B did not drop the original packet, this condition would hold.

Third, we assume each node has a public key certificate signed by a trusted certificate authority (CA) and also an authentic public key of the CA. Our protocol relies on these public keys to bootstrap trust in the ad hoc network. We believe this a reasonable assumption if all the nodes in the network are from the same autonomous system (AS) or administrative unit. The distribution of certificates and keys can be done in any reliable way. An advantage of using public key certificate is its flexibility and scalability; a node can authenticate other nodes independently, and there is no need for them to have pre-shared keys as in symmetric cryptography. We shall discuss the mechanisms other researchers have proposed for bootstrapping trust in Section 7.

Fourth, we assume that the mobile nodes under consideration are relatively underpowered. Public-key operations such as digital signatures are relatively expensive to compute on platforms such as handhold PDAs. Brown *et al* [1] have measured the speed of RSA signing and verifying on various platforms. For example, a 512-bit RSA signature generation takes $2 - 6$ seconds on a RIM Pager and a Palm Pilot, whereas a signature verification takes $100 - 200$ milliseconds when the public exponent $e = 3$.

We also assume loose time synchronization in the ad hoc network so that we can utilize TESLA. The security of TESLA in has been analysed in previous work [10, 25, 26].

3.2 LHAP Overview

The main goal of our protocol is to provide network access control, i.e., to prevent unauthorized nodes from being able to inject traffic into the ad hoc network. To achieve this goal, under LHAP, every node in the network authenticates every packet it receives from its neighbors before forwarding it (if necessary). Packets from unauthorized nodes are dropped, thus preventing them from propagating through the network. A packet that needs multiple hops before reaching its destination thus gets authenticated by each node on its path. We refer to this as *hop-by-hop* authentication.

LHAP's efficiency gains over traditional authentication protocols derive from two techniques: (i) lightweight packet authentication, and (ii) lightweight trust management. Since all packets are authenticated on every hop on their paths, it is necessary that the packet authentication technique used by LHAP be as inexpensive as possible. LHAP employs a packet authentication technique based on the use of one-way hash chains. Secondly, LHAP uses TESLA to reduce the number of public key operations for bootstrapping trust between nodes, and also use TESLA for maintaining the trust between nodes. We briefly discuss both these issues below before discussing the operations of the protocol in more detail.

Lightweight Traffic Authentication Since we propose to authenticate all traffic packets, we must use a computationally inexpensive technique that can also provide immediate authentication. Like TESLA, the traffic authentication technique used by LHAP is based upon the use of one-way key chains. Unlike TESLA, however, our authentication technique does not use periodic and delayed key disclosure. Delayed authentication (as in TESLA) is not appropriate for LHAP since a packet would be delayed at each node in the path from the source to the destination. For example, a traffic packet sent to another node ten hops away will take at least ten seconds if we use TESLA for authentication and the key disclosure period in TESLA is one second. Moreover, since each node has to buffer the traffic packets it has received until they are authenticated, delayed authentication will lead to large storage requirements at every node. We note the variant of TESLA proposed in [25] does not solve this problem either because every node is both a sender and a receiver in the network.

In LHAP, each node that is source of packets generates a one-way key chain that is used for traffic authentication by its immediate neighbors. We use the term TRAFFIC key to refer to the keys in this one-way key chain. Every neighbor of a node will obtain an authentic key in this TRAFFIC chain when it first establishes a trust relationship with the node. A node transmitting a packet will append a new TRAFFIC key to the packet. All the neighboring nodes receiving this packet can verify the authenticity of this packet by checking the validity of the attached TRAFFIC key.

Trust Management Nodes can bootstrap their trust relationship, i.e., exchange authentic TRAFFIC keys, by using a public key based technique. A simple approach that can be used is one in which a node signs its most recently released TRAFFIC key and sends it to every new neighbor. However, this approach does not scale well for resource-constrained mobile nodes which may encounter a large number of nodes during their lifetime. To address this issue, LHAP use TESLA to reduce the number of signature operations. Specifically, in LHAP every node only uses digital signatures to bootstrap a TESLA key chain. Subsequently, TESLA keys are then used to provide authentic TRAFFIC keys.

To maintain the trust relationship, a node broadcasts its latest released TRAFFIC keys periodically, authenticated by its TESLA keys. Thus its neighbors will drop the replayed packets authenticated by any earlier TRAFFIC keys. We call this periodic message a KEYUPDATE message.

When a node does not receive a valid KEYUPDATE message from a neighbor within a TESLA interval (mostly because of being out of transmission range), it will terminate its trust of this neighbor. If the two nodes come within transmission range, they will need to run the trust bootstrapping process to reestablish a trust relationship.

3.3 Notation

We use the following notation to describe security protocols and cryptography operations in this paper:

– A, B are principals, the identities of mobile nodes.
– $Cert_A$ is node A's public-key certificate issued by a trusted CA.
– $Sign_A(M)$ denotes the digital signature of message M, signed with node A's private key.
– $M1|M2$ denotes the concatenation of message $M1$ and $M2$.
– $MAC(K, M)$ denotes the computation of MAC over message M with key K.
– $K_A^T(i)$ denotes node A's i'th key in its TESLA key chain, while $K_A^F(i)$ denotes its i'th key in its TRAFFIC key chain.

3.4 LHAP in Detail

LHAP consists of two security building blocks: *traffic authentication* and *trust management*. We discuss these topics in more detail below.

Traffic Authentication The goal of LHAP is to provide full network access control, As such, LHAP does not distinguish between data packets and routing control packets for authentication purposes. LHAP is transparent to and independent of the network routing protocol. It can be thought of as residing between the data link layer and the network layer, providing a protection mechanism that can prevent or thwart many attacks from happening. This transparency and independence allows LHAP to be turned on or off without affecting the operations of other layers.

Every node uses a TRAFFIC key chain for authenticating the traffic packets originating from itself or received from its neighbors for forwarding. For example, consider a node A that wants to broadcast a packet M. Let its next TRAFFIC key be $K_A^F(i)$. It will send the following message

$$A \longrightarrow * : M, K_A^F(i). \tag{1}$$

Every receiving node verifies the authenticity of this packet by verifying the TRAFFIC key $K_A^F(i)$, based on the most recent TRAFFIC key, $K_A^F(j), j < i$, that it received from node A. It will then replace $K_A^F(j)$ with $K_A^F(i)$ in its record. In LHAP, a node only authenticates traffic packets from its direct neighbors, thus it is very difficult for an attacker to launch replay attacks, as we discussed in Section 3.1.

Using TRAFFIC key for traffic authentication has the following benefits. First, it enables instant verification of traffic packets. Second, unlike TESLA keys, it is not necessary to disclose TRAFFIC keys periodically; disclosing keys perodically would result in a severe wastage of keys when a node has no packets to transmit. Therefore, in LHAP the rate at which a node consumes its TRAFFIC keys can be adapted to the actual traffic rate. Third, it is computationally more efficient than computing HMAC over the entire message, because it only requires computing a hash over a key of a small fixed size (e.g., 8 bytes). However, we also note this scheme does not achieve the same level of security as in TESLA, as a tradeoff between security and performance. We shall discuss the possible attacks on TRAFFIC keys in Section 4.

Trust Management Trust management includes *trust bootstrapping*, *trust maintenance* and *trust termination*.

Trust Bootstrapping When a node wants to join an ad hoc network, it first pre-computes a one-way key chain and a TESLA key chain. Then it signs the commitments of these key chains, and broadcasts them to its neighbors. In Fig. 1, we show a scenario where node A starts to join a network where its neighbors are B,C,D and E. A broadcasts a JOIN message with $TTL = 1$.

$$A \longrightarrow * : Cert_A, Sign_A\{A|K_A^T(0)|K_A^F(0)|T_A^T(0)|T_A^F(0)\}, \tag{2}$$

where $T_A^T(0)$ and $T_A^F(0)$ are the starting times for its TESLA and TRAFFIC key chains respectively. Every receiving node first verifies the authenticity of node A's certificate using the CA's public key, then uses node A's public key in the certificate to verify the signature on the message. It will record the commitments of node A's key chains and their starting times if all the verifications are successful. To bootstrap an authentic TRAFFIC key and a TESLA key to node

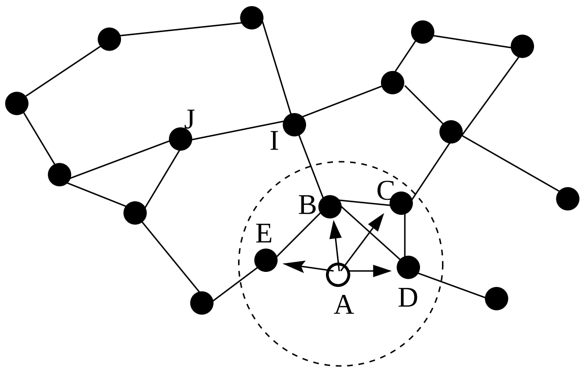

Fig. 1. A scenario where node A joins the ad hoc network

A, each of its neighbors (say B) unicasts the following ACK message to A:

$$B \longrightarrow A : Cert_B, Sign_B\{B|K_B^T(0)|K_B^F(0)|T_B^T(0)|T_B^F(0)\}, MAC(K_B^T(i), K_B^F(j)), \tag{3}$$

where the signature part was generated when node B first joined the network, $K_B^F(j)$ is node B's most recently released TRAFFIC key, and $K_B^T(i)$ is node B's next TESLA key to be released. After receiving this message, node A first does two verifications to obtain node B's authentic key chain commitments, as its neighbors did to verify its JOIN message, and then record them. Node A cannot verify the MAC until node B releases $K_B^T(i)$. The disclosure delay is half of a TESLA interval on average. Note that the variant of TESLA [25] cannot be applied here to provide immediate authentication of $K_B^F(j)$ because a node cannot predict how many TRAFFIC keys it will consume in the next TESLA interval. After it receives and verifies $K_B^T(i)$ later, node A updates $T_B^T(0)$ and $T_B^F(0)$ to $K_B^T(i)$ and $K_B^F(j)$ respectively for its record, and starts to forward valid traffic from node B, as described in Section 3.4.1.

Trust Maintenance Periodically, each node broadcasts an KEYUPDATE message (with TTL=1) to its neighbors, which contains its most recently disclosed TRAFFIC key. The KEYUPDATE message is authenticated with the next TESLA key in its key chain. As an example, the KEYUPDATE message node A sends is

$$A \longrightarrow * : A, K_A^T(i-1), MAC(K_A^T(i), K_A^F(j)), \tag{4}$$

where $K_A^F(j)$ is node A's most recently released TRAFFIC key, and $K_A^T(i)$ is node A's next TESLA key to be released. In addition, node A includes $K_A^T(i-1)$ to allow its neighbors to verify the previous KEYUPDATE messages from node A. The purpose of broadcasting KEYUPDATE messages is for preventing malicious nodes from forging traffic using the TRAFFIC keys node A has already released.

There are two possible actions that a node can take when it receives the above KEYUPDATE message. It can opt to not forward any traffic from node A until it receives the TESLA key for authenticating the MAC. Alternatively, it can forward immediately the traffic packets which contain valid TRAFFIC keys (i.e., verified to be correct and also later in node A's TRAFFIC key chain than the key in the KEYUPDATE message) even though it has not yet verifed the MAC. The first approach, obviously, has severe impact on the performance of the upper layer protocols (e.g. the routing protocol) because a node may drop many traffic packets from other nodes. Therefore, LHAP uses the second approach instead. We shall discuss the security of this approach in more detail in Section 4.

Trust Termination In LHAP, there are two scenarios under which the trust relationship between nodes will be terminated. First, when a compromised node is detected, all the nodes will terminate their trust relationship with that node permanently. Second, when a node does not receive a (valid) KEYUPDATE message from a neighbor within a TESLA interval because the latter has moved out of its transmission range, it will terminate its trust of this neighbor temporarily. If the two nodes move within transmission range again, they can run the trust bootstrapping process again to reestablish their trust relationship, if they do not have any cached commitments of the other's key chain. Otherwise, they can reestablish their trust relationship using TESLA, with a delay of half a TESLA interval on average.

4 Security Analysis

In this section, we discuss some possible attacks against LHAP, which are all against traffic key chains. We assume TESLA is secure, given loose synchronization in the network.

4.1 Outsider Attacks

Outsider attacks are attacks launched by nodes that do not possess a valid certificate. We identify three types of outsider attacks here.

Single Outsider Attack In Fig. 1, we showed a situation where node E received node A's JOIN message when node A joined the network. From the JOIN message, node E obtained the authentic commitments of node A's key chains. Now we consider a scenario in Fig. 2 where node E has moved out of node A's transmission range for a time period (say many TESLA intervals). During the time, node A has disclosed many of its TESLA keys and TRAFFIC keys. An outside attacker, node P_2, may eavesdrop and use these keys to impersonate node A. For instance, suppose node A has broadcast a packet with content M and $K_A^F(i)$. Node P_2 may modify M to M' with the same TRAFFIC key, and sends it to node E. Because TRAFFIC keys are not periodically disclosed, node E cannot determine which TRAFFIC key node A is using.

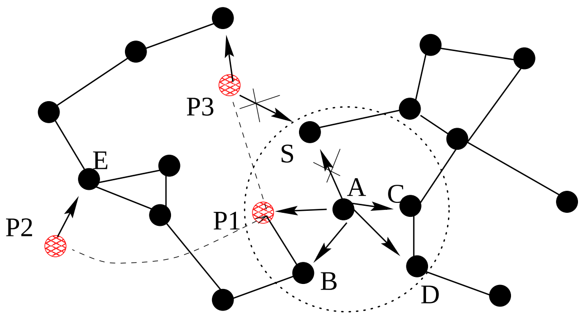

Fig. 2. Various attacks on LHAP. P1, P2 and P3 are malicious nodes, and the dashed lines between them are private channels

To thwart this attack, we designed the *trust termination* phase as in Section 3.4.4. Since node E has not heard from node A for a time period of more than one TESLA interval, it will not forward any traffic from node P_2 until it receives a valid KEYUPDATE message. Since TESLA keys are disclosed periodically, node E knows which TESLA keys node A has released. Therefore, node P_2 cannot use the TESLA keys that node A previously disclosed to forge an KEYUPDATE message. On the other hand, node P_2 cannot forge a valid KEYUPDATE message using any TESLA keys that have not been released yet by node A due to the one-wayness of pseudo random functions.

Collaborative Outsider Attack A *collaborative outsider attack* (also called a *wormhole* attack [11]) is launched by multiple colluding outside attackers. In Fig. 2, the attackers P_1 and P_2 have a private channel that allows them to communicate directly. P_1 forwards every message it eavesdropped from node A, including KEYUPDATE messages and traffic packets, to P_2 through the *wormhole*. P_2 then rebroadcasts the KEYUPDATE messages and modify the traffic packets to deceive node E. Due to time synchronization errors, node E may still accept the replayed KEYUPDATE messages, and will forward the modified traffic packets for node P_2.

This attack can be detected if the mobile nodes carry devices such as Global Positioning Systems (GPS). A node can put its GPS coordinates in its KEYUPDATE messages to allow a receiving node to determine if they should hear each other. For example, in Fig. 2 node E and A should not hear each other based on their coordinates; therefore, node E will detect the inconsistency of node A's position when it receives the replayed KEYUPDATE messages from P_2, and therefore drop the later received messages from P_2. We note our protocol does not address this attack completely. Due to the one-TESLA-interval delay for verifying an KEYUPDATE message, node E will still forward the (probably modified) traffic packets from P_2 in this interval if the packets carry TRAFFIC keys that are verified to be correct and later than the one in the KEYUPDATE message in node A's TRAFFIC key chain. However, there is an upper bound on the number of forged packets, which is the number of packets node A actually sends in this interval, because an attacker cannot compute the TRAFFIC keys which node A has not disclosed yet.

In case that a node has no GPS equipped, a less secure approach involves using neighbor information[1]. A node includes a list of its neighbors in its KEYUPDATE message. A receiving node, say node E in Fig. 2, will be alerted if it finds that a KEYUPDATE message from node P_2 (impersonating node A) does not include node E as a neighbor. We note if node P_2 also forwards every message from node E to node P_1 via the *wormhole* so that node P_1 can impersonate node E to node A, then both node A and E will include each other in their neighbor lists. However, for ad hoc networks with high degree of connection, this attack will be very difficult to launch without being detected because two neighboring nodes generally should have many common neighbors.

Hidden Terminal Attack A *hidden terminal attack* on TRAFFIC keys is more subtle, and it is motivated by the hidden-terminal problem in mobile networking.

In TDMA (time division multiple access) networks, CSMA (carrier sense multiple access) protocols have been used to avoid collision for scenarios where multiple nodes share a common channel. These protocols require each node to listen to the channel before it transmits to prevent other nodes within its transmission range from transmitting simultaneously. But it does not solve the problem when two nodes that cannot hear each other sends to a common neighbor simultaneously; as a result, the common node will drop both packets on detection of collision. This is called hidden-terminal problem in wireless networking. IEEE 802.11 solves this problem using CSMA/CA with ACKs and optional RTS/CTS control packets. For this scheme to work, however, an assumption that these contenting nodes will cooperate has to be made. In Fig. 2 we show an attack that tries to disrupt this cooperation, which we call a hidden-terminal attack.

Suppose node A is broadcasting a traffic packet that encloses a TRAFFIC key $K_A^F(j)$ for packet authentication. To attack node S, a malicious node P_3 transmits a packet to node S at the same time, which causes node S to drop both packets. Meanwhile, node P_1 can send $K_A^F(j)$ to node P_3 through a *wormhole*. Now node P_3 can send an erroneous packet to node S impersonating node A using $K_A^F(j)$ before node A does a retransmission; consequently, node S will drop the retransmitted authentic packet from node A.

However, since the retransmission interval is usually very small (tens of microseconds), the attackers may have to run continuous attacks to prevent node A's retransmitted message from successfully being received. This can be easily detected because the hidden-terminal problem does not happen that frequently in a network where the RTS/CTS control packets are deployed. In addition, impersonating a node within a range of two hops is very likely to be detected by other nodes.

4.2 Insider Attacks

Insider attacks are attacks launched by one or more compromised nodes that possess valid certificates. We identify three possible insider attacks.

[1] Neighborhood knowledge could be inferred from the Medium Access Control layer or from the traffic packets

Single Insider Attack A compromised node might attempt to flood the network with many traffic packets. For schemes which provide source authentication, having an upper bound on traffic rate could limit this attack. However, our hop-by-hop authentication scheme does not provide strong source authentication, because every node only authenticates its neighbors instead of the original traffic sources for the purpose of scalability. Thus, a compromised node might broadcast malicious traffic while pretending to be a forwarding node.

Insider Clone Attack A *clone attack* occurs when a compromised node shares its private key (hence its identity) with its outside conspirators. Due to their having the same identity, these nodes are less likely to launch collaborative attacks without being detected. The cloned nodes are therefore more likely to be distributed in different locations of the network. Indeed, this clone attack can be considered as multiple independent *single insider attack*s.

Multiple-insider Attack This attack is launched by multiple compromised insiders, each of which holding a legitimate certificate. Coalition of these insiders could result in very sophisticated attacks.

Generally, it is more difficult to detect the attacks launched by insider nodes, especially by multiple collaborative nodes. LHAP alone does not have complete solutions for addressing these attacks, although the use some techniques might mitigate the severity. For instance, in the *single insider attack*, the neighbors could be aware of the attack if a node pretends to be a forwarder for the packets originated from itself. We note a better solution is that every node is installed with an intrusion detection system (IDS) which collects trace data imported from all the layers, because compromised nodes could launch attacks against multiple layers, such as routing layer and application layer. Moreover, multiple nodes could also perform cooperative detection. For example, in the *insider clone attack*, after exchanging its trace data with another node E, a node A might detect the attack if they both have met a third node P at about the same time but at quite different locations (if GPS is equipped). Zhang and Lee [34], Marti et al. [21] have studied the intrusion and misbehavior detection issue in mobile networks.

Finally, after detecting the attacks and identify the compromised nodes, one or a certain number of nodes[2] each sign a node revocation notice that include the identities of the compromised nodes and its public certificate, and flood the message in the network. All the receiving nodes then add the compromised nodes into their local revoked node lists (RNL). As a result, they drop all the traffic from these compromised nodes in the future.

5 Performance Analysis

We mainly consider the following performance metrics in LHAP.

– **Computational Overhead** LHAP introduces two types of computational overhead, in traffic authentication and in trust management respectively. The computational overhead for verifying a traffic packet is usually very small due to the efficiency of a hash function, although a node might need to compute multiple hashes to verify a packet in some cases. The computational overhead for trust management mainly includes a digital signature for bootstrapping its key chains, two signature verifications for every JOIN message from its newly encountered neighbors, and one or several hash computations for every KEYUPDATE message. The digital signature is the most expensive operation here, but it is only performed once. On the other hand, signature verification is several order slower than

[2] Using a threshold based scheme can prevent up to t malicious nodes from revoking a legitimate node

a hash computation; therefore, the computational overhead is mainly affected by the number of signature verifications which is determined by the number of encountered nodes. The number of encountered nodes is instead mainly determined by network density and node mobility. Obviously, a node encounters more nodes in a network with a higher node density and higher node moving velocities. Generally, LHAP is computationally scalable because the overhead of verifying signatures is affordable even for the handhold PDAs [1] that have very constrained computational capability.

– **Latency** In LHAP, a node verifies a traffic packet it receives by computing one or several hashes. Thus the additional latency LHAP incurs is usually neglectable with respect to the overall normal end-to-end transmission latency of a packet.

– **Traffic Byte Overhead** We define traffic byte overhead as the number of all the non-traffic bytes a node transmits per time unit. There are four sources of traffic byte overhead in LHAP. First, a node adds a traffic key to every traffic packet it sends; hence, the overhead is one key per traffic packet and the overall overhead is mainly determined by the number of data sources and their traffic models. Second, a node sends a JOIN message at the time it joins the network, and the overhead for a JOIN message is determined by the size of a public key certificate and the size of a digital signature. Third, a node sends an ACK packet to every new neighbor, and an ACK packet is one key size and one MAC size larger than a JOIN message. The overall overhead, like the computational overhead for signature verification, is also determined by network density and node mobility. Fourth, a node periodically sends a KEYUPDATE message (that includes two keys one MAC), and its overhead depends on the TESLA interval. The greater the TESLA interval, the smaller the overhead. We note the byte overhead in the periodic KEYUPDATE messages and the traffic packets accounts for a major fraction of the overall byte overhead, because other types of byte overhead can be amortized during the lifetime of a node.

  Example Let us assume we use 10 bytes for a key (including a 2 bytes key id), 10 bytes for a MAC, 500 bytes on average for a traffic packet, 256 bytes for a public key certificate, 128 bytes (1024 bits RSA) for a digital signature. Suppose a node joins the network for one hour, during which it sends (or forwards) 1000 packets and encounters 100 nodes. If the TESLA interval is 1 second, the traffic byte overhead is 44 bytes/s; If we use 2 second for the TESLA interval, the traffic byte overhead is 29 bytes/s. We believe this amount of overhead is reasonable for a security service.

– **Traffic Delivery Ratio** We define this ratio as the fraction of traffic packets that a node accepts in all the packets it receives from its neighbors. In LHAP, a node might drop traffic packets that are from legitimate neighbors in two scenarios. The first scenario is when it encounters a neighbor at the first time, and the second scenario is when it re-encounters a neighbor after more than one TESLA interval since their last encounter. In both cases, it will drop packets from this neighbor if the neighbor is broadcasting[3] packets, e.g., ROUTING REQUEST packets in DSR [14] or data packets, until it receives valid ACK message or KEYUPDATE message from this neighbor in at most one TESLA interval. Therefore, the traffic delivery ratio is mainly affected by the TESLA interval, traffic rate and node mobility model. The greater the TESLA interval, the more traffic packets likely being dropped, hence the lower the traffic delivery ratio. On the other hand, if the frequency of a node's re-encounters with other nodes

[3] A node usually does not send a unicast packet to another node that just becomes its neighbor in most routing protocols

becomes higher, the traffic delivery ratio will also be lower. However, our detailed simulation using the Network Simulator (ns2) [22], which uses similar parameter settings as in [10], shows that in most cases the traffic delivery ratio in LHAP is greater than 99.9%. Moreover, we note that one or several nodes occasionally dropping broadcast packets has very little effect in the delivery ratio of the application data due to the flooding nature of the broadcast packets.

The above performance analysis shows LHAP is a light-weight security protocol on both computation and communication, and yet has very small affect on the performance of other layer protocols.

6 Issues In Deployment

We discuss the issues related to the deployment of LHAP in this section.

6.1 Interaction With Routing Protocols

LHAP is independent of and resides under the network layer protocols. In practice, it could take advantage of the deployed network routing protocol to achieve better efficiency. Some of the ad hoc routing protocols in literature, e.g., AODV, TORA, AMRIS [31] *et al*, require nodes periodically exchange routing information or beacon messages with their neighbors. Thus LHAP can piggyback its KEYUPDATE messages in these messages to avoid transmitting separate KEYUPDATE packets, although both the bandwidth and the energy for transmission and receiving are not much reduced, if any. Note this combination does not affect the transparency of LHAP with respect to the routing protocol, because the LHAP agent in a receiving node removes the piggybacked KEYUPDATE message prior to submitting the message to the network layer.

6.2 Supporting Very Long Key Chains

In LHAP, TESLA keys are disclosed periodically. For a network with a lifetime of five hours and TESLA interval of one second, the TESLA key chain will have a length of $5 * 3600 = 18,000$ keys. On the other hand, the TRAFFIC keys are usually consumed at a much higher rate, depending on the application. As a result, very long key chains are required in LHAP.

If a node stores all the keys in a key chain, it can disclose any keys immediately on demand. Obviously, this approach is not scalable in terms of its memory requirements. Another scheme is one in which a node only keeps the last key of the key chain, and every time computes the key to be disclosed from the last key as in the key chain generation. However, this scheme is not scalable in terms of computational costs. Therefore, the size of a key chain impacts both memory requirements and computation costs.

To address this issue, Coppersmith and Jakobsson [4] presents an optimization algorithm that can be used to trade storage and computation costs. Their algorithm performs $\lceil d \log_2 N \rceil$ hashes per output element, and uses $\lceil d \log_2 N \rceil$ memory cells, where the size of each cell is slightly larger than that of a key and N is the length of the key chain.

The multilevel key chain scheme proposed by Liu and Ning [18] can be used to further reduce the overhead on storage and computation. The main idea is to have multiple key chains of different disclosure intervals. As an example, we show how to design a two-level scheme for a key chain of length 3600 in LHAP. A node A first generates a high-level key chain of length 60, denoted as $\langle K^h_A(i)\rangle, i = 0, ..., 59$, using a pseudo random function F_1. For each $K^h_A(i)$, node A computes a seed $K^l_A(i) = F_2(K^h_A(i))$, where F_2 is a second pseudo random function. Node A then generates a low-level key chain of length 60, denoted as $\langle K^l_A(i,j)\rangle, j = 0, ..., 59$, using $K^l_A(i)(i \neq 0)$ as the last key (i.e., $K^l_A(i) = K^l_A(i, 59)$) and a third pseudo random function F_3. In its JOIN message, node A signs $K^h_A(0)$ and $K^l_A(1, 0)$ instead of $K^T_A(0)$. When all the keys in a low-level key chain $\langle K^l_A(i,j)\rangle$ are disclosed, node A then discloses the key $K^h_A(i)$ that generated $\langle K^l_A(i,j)\rangle$. Then the keys in the key chain $\langle K^l_A(i+1,j)\rangle$ are disclosed, and so on. One advantage of this scheme is it allows a low-level key chain to be generated on the fly. Note both the high-level and the low-level key chains can use the optimization scheme in [4] to trade off between storage and computation.

6.3 Supporting Fast Verification

Consider the following two scenarios. One is illustrated in Fig. 2, where node E only has node A's first TESLA key which it obtained at the time of node A's join. It moved out of the transmission range of node A immediately after the join. Suppose the TESLA interval is one second, then node A has disclosed 3600 keys when it encounters node E again one hour later. Thus node E has to compute 3600 hashes to verify the correctness of node A's current TESLA key. The second scenario is similar. When a node joins the network much later, it might also have to compute a large number of hashes to verify neighboring nodes which joined much earlier.

Although computation of hash functions is very efficient, it is still undesirable to have such large number of computations. On the other hand, an attacker may utilize this feature to consume a node's computation resources by engaging it in doing a large number of hash computations. So in this case, a fast verification algorithm is needed to reduce the verification overhead.

The tree-based authentication scheme, also known as Merkle hash tree [20] and further studied in [11], can be deployed here to assure the maximum number of verifications a receiver has to perform is $O(log(N))$, where N is the length of a TESLA key chain. The key idea is to construct a binary hash tree from the TESLA key chain. More specifically, every leaf key k'_i in the tree, from the leftmost to the rightmost, is computed from a distinct key k_i in the TESLA key chain, from the first commitment to the last one, using a pseudo-random function F_1 such as $k'_i = F_1(k_i)$, An intermediate key or the root key k_j in the hash tree is computed from its children key k^l_j and k^r_j using another pseudo-random function F_2 such as $k_j = F_2(k^l_j|k^r_j)$. When a mobile node, say A, joins a network, besides the first commitment of its TESLA key chain, it also provides the authentic root key in its hash tree. To allow another node, say B, to verify a key k_i in its key chain, node A sends all the keys that are the siblings to the keys in k_i's path to the root of the tree. Node B then verifies k_i by verifying if it can compute a root key that is the same as the one in node A's signature, based on all the keys it received. Thus, in this algorithm, B only needs to compute $O(log(N))$ hashes.

Note that the above verification algorithm is only used for TESLA key chains, not for TRAFFIC key chains. Because the most recently disclosed TRAFFIC keys are broadcast in the KEYUPDATE messages in LHAP, the number of hashes a node has to compute to verify a TRAFFIC key is bound by the number of traffic packets transmitted in one TESLA interval.

7 Related Work

Previous work in ad hoc network security roughly falls into the following categories: trust and key management, secure routing and intrusion detection.

Stajano and Anderson [30], Balfanz et al. [3] propose to establish trust and keys between nodes through physical contact in the absence of an online authentication server. Zhou and Hass [33] propose to use threshold signature schemes to prevent one or a few compromised nodes from signing messages for the group, and later Kong et al. [16] extend this approach to a fully distributed way. LHAP can use their threshold schemes for node revocations. Unlike the above approaches that are mainly based on public key techniques, Hu, Perrig and Johnson [10] also propose to use an online trusted Key Distribution Center (KDC) to help establish trust relationship between pairs of nodes.

Other work is related to network key management. To setup a secret key for a pair of nodes, one approach is to preload each of them with the key, but it requires a node to store $N-1$ preloaded keys for a network with N nodes, for a total of $N(N-1)/2$ keys. Eschenauer and Gligor [6] proposed a more efficient probabilistic key sharing scheme to reduce the number of keys a node has to store for sensor networks. Basagni et al [2] proposed a clustering scheme for group re-keying in sensor networks where sensor nodes are assumed unbreakable.

Secure routing for ad hoc networks has been extensively studied recently. Dahill et al [5] identify several security vulnerabilities in AODV and DSR, and proposed to use asymmetric cryptography for securing ad hoc routing protocols. Although their approach could provide strong security, performing a digital signature on every routing control packet could lead to performance bottleneck on both bandwidth and computation. Yi, Naldurg, and Kravets [32] present a security-aware routing protocol which uses security (e.g., trust level in a trust hierarchy) as the metric for route discovery between pairs. Perrig et al. [28] use symmetric primitives for securing routes between nodes and a trusted base station in a resource extremely constrained sensor network. Papadimitratos and Hass [27] propose a routing discovery protocol that assumes a security association (SA) between a source and a destination, whereas the intermediate nodes are not authenticated.

Hu, Perrig and Johnson designed SEAD [9] which uses one-way hash chains for securing DSDV, and Ariadne [10] which uses TESLA and HMAC for securing DSR. In LHAP, we also utilize these techniques due to their efficiency. The main difference between LHAP and their protocols is in the design goals. Their protocols are designed for securing *specific routing* protocols, while we design LHAP as a *general network access control* protocol which provides authentication for every packet and is independent of the routing protocols.

Other work in ad hoc network security include work on intrusion and misbehavior detection. Zhang and Lee [34] describe several intrusion detection and response mechanisms for ad-hoc networks. Marti et al [21] consider the problem of detecting selfish intermediate nodes that do not forward packets.

8 Conclusions

In this paper, we have presented LHAP, a lightweight hop-by-hop authentication protocol for network access control in ad hoc networks. LHAP is based on two techniques: (i) hop-by-hop authentication for verifying the authenticity of all the packets transmitted in the network and (ii) one-way key chain and TESLA for packet authentication and for reducing the overhead for establishing trust among nodes. The design of LHAP is transparent to and independent of the routing protocols. Through our security and performance analysis, we show LHAP is beneficial and also practical.

References

1. M. Brown, D. Cheung, D. Hankerson, J. Hernandez, M. Kirkup, and A. Menezes. PGP in Constrained Wireless Devices. In 9th USENIX Security Symposium, pages 247261, August 2000.
2. S. Basagni, K. Herrin, E. Rosti, D. Bruschi, Secure Pebblenets, MobiHoc 2001
3. D. Balfanz, D. Smetters, P.Stewart and H. Wong. Talking to Strangers: Authentication in Ad-Hoc Wireless Networks. In Symposium on Network and Distributed Systems Security (NDSS'02), 2002.
4. D. Coppersmith, M. Jakobsson, Almost Optimal Hash Sequence Traversal, Finanical Cryptography (FC) 02
5. B. Dahill, B. Levine, E. Royer, C. Shields, A Secure Routing Protocol for Ad-Hoc Networks, Technical Report, UMass-CS-2001-037.
6. L. Eschenauer and V. Gligor. A Key Management Scheme for Distributed Sensor Networks. In ACM CCS2002, Washington D.C., 2002.
7. O. Goldreich, S. Goldwasser, and S. Micali, How to Construct Random Functions, Journal of the ACM, vol. 33, no. 4, 1986, pp 210-217.
8. J. Hubaux, L. Buttyan, S. Capkun. The Quest for Security in Mobile Ad Hoc Networks. MobicHoc 2001.
9. Y. Hu, D. Johnson, A. Perrig. SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks. Proceedings of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002), IEEE, Calicoon, NY, June 2002. .
10. Y. Hu, A. Perrig, D. Johnson. Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks. in Mobicom 2002
11. Y. Hu, A. Perrig, and D. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks. Proceedings of INFOCOM 2003, IEEE, San Francisco, CA, April 2003, to appear.
12. IEEE Computer Society LAN MAN Standards Committee. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11, 1997
13. D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In T.Imielinski and H. Korth, editors, Mobile Computing , Pages 153-181. Kluwer Adcademic Publishers, 1996.
14. D. Johnson, D. Maltz, Y. Hu, J.Jetcheva. The Dynamic Souce Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft-ietf-manet-dsr-07.txt, February 2002.
15. Vesa Karpijoki. Signaling and Routing Security in Mobile and Ad-hoc Networks. http://www.hut.fi/ vkarpijo/iwork00/.
16. J. Kong, P. Zerfos, H. Luo, S. Lu, L. Zhang. Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks, Ninth Internation Conference on Network Protocols (ICNP'01)
17. Leslie Lamport, Password authentication with insecure communication communication. Communications of the ACM, 24(11):770-772, Nov., 1981.
18. D. Liu and P. Ning, Efficient Distributed of Key Chain Commitments for Broadcast Authentication in Distributed Sensor Networks To appear in NDSS'03.
19. S. Maki, T. Aura, M. Hietalahti, Robust Membership Management for Ad-hoc Groups, in Proc. 5th Nordic Workshop on Secure IT Systems (NORDSEC 2000).
20. Ralph Merkle. A certified digital signature. In Gilles Brassard, editor, Advances in Crypto-89, pages 218-238, Berlin, 1989.
21. S. Marti, T. Giuli, K. Lai, M. Baker. Mitigating Routing Misbehavior in Mobile Ad Hoc Networks. ACM MOBICOM, 2000

22. NS-2, http://www.isi.edu/nsnam/ns/index.html
23. V. Park and M. Corson, A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks, Proceedings of IEEE INFOCOM '97, Kobe, Japan (April 1997)
24. C. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In Proceedings of SIGCOMM'94. 1994.
25. A. Perrig, R. Canetti, D. Song, and J. Tygar. Efficient and secuire source authentication for multicast. In Network and Distributed System Security Symposium, NDSS'01, Feb. 2001.
26. A. Perrig, R. Canetti, J. Tygar, D. Song. Efficient authentication and signing of multicast streams over lossy channels. In IEEE Symposium on Security and Privacy. May 2000.
27. P. Papadimitratos and Z. Haas. Secure Routing for Mobile Ad hoc Networks. In SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002), 2002.
28. A. Perrig, R. Szewczyk, V. Wen, D. culler, and J. Tygar. SPINS: Security Protocols for Sensor Networks. In Seventh Annual ACM International Conference on Mobile Computing and Networks(Mobicom 2001), Rome Italy, July 2001.
29. Charles Perkins. Ad hoc On Demand Distance Vector (AODV) Routing, Internet draft, draft-ietf-manet-aodv-00.txt.
30. F. Stajano and R. Anderson. The Resurrecting Ducking: Security Issues for Ad-hoc Wireless Networks. In Security Protocols, 7th International Workshop, 1999.
31. C. Wu, Y. Tay, and C. Toh, Ad hoc Multicast Routing protocol utilizing Increasing id-numberS (AMRIS) Functional Specification, Internet-Draft, draft-ietf-manet-amris-spec-00.txt, Nov. 1998, Work in progress.
32. S. Yi, P. Naldurg, R. Kravets. Security-Aware Ad-Hoc Routing for Wireless Networks. Technical Report, UIUCDCS-R-2001-2241, UILU-ENG-2001-1748
33. L. Zhou and Z. Hass. Securing Ad Hoc Networks. IEEE Network Maganize, 13(6), November/December 1999.
34. Y.Zhang and W.Lee. Intrusion Detection in Wireless Ad-Hoc Networks. MOBICOM 2000.