A Lightweight Authentication Protocol for Mobile Ad Hoc Networks

Bin Lu, and Udo W. Pooch

Department of Computer Science
Texas A&M University
College Station, TX 77843-3112, USA

{bin4549, pooch}@cs.tamu.edu

Abstract

The characteristics of mobile ad hoc networks (MANETs) determine that the authentication approaches to protect routing and data packet transmission in MANETs should be lightweight and scalable. In this paper, we propose a lightweight authentication protocol, which utilizes one-way hash chain to provide effective and efficient authentication for communications between neighboring nodes in MANETs. Delayed key disclosure scheme is used to prevent from in-the-middle attack on key release. The security properties of the protocol are analyzed in the paper. We also demonstrate simulation results and performance analysis on trust management, message authentication and the delayed key disclosure approach. The analysis shows that the protocol incurs low overhead penalty and achieves a low dropped packet rate on key disclosure with a cache of fair size.

Keyword: MANET, security, authentication, one-way hash function.

I. Introduction

Authentication mechanisms are used to ensure that the entity who supposedly sent a message to another party is indeed the legitimate entity. General security requirements for authentication include protection against replay attacks, resistance against man-in-the-middle attacks and provision of confidentiality. There are two basic kinds of cryptography that have been widely used for the traditional Internet: *symmetric* cryptography and *asymmetric* cryptography (such as digital signature).

Different from the fixed networks, the communication links in mobile ad hoc networks are open shared medium, which makes the communications between neighboring nodes more vulnerable to attacks such as packet forging and malicious alteration. In addition, mobile ad hoc networks are characterized by absence of fixed infrastructure, rapid topology change and constrained resources (such as limited battery power, small computational capacity and bandwidth). These characteristics determine that the authentication protocols used for routing and data packet delivery in mobile ad hoc networks (MANETs) should be lightweight and scalable. Asymmetric cryptography does not adapt well to MANETs in that the processing required for asymmetric cryptography is very CPU (Central Process Unit) intensive and the technique has been proved to be prohibitively insufficient in wireless ad hoc networks in terms of message overhead and computation complexity. Symmetric

cryptography algorithms are fast. Nevertheless, it introduces complexity in key maintenance and exerts difficulty in authentication for multicast or broadcast communications.

Moreover, radio channels in wireless networks are more erroneous and lossy than the communication links in the Internet. With multiple receivers, there could be a high variance among the bandwidth and radio interference of different receivers, with high packet loss for the receivers with low bandwidth and high radio interference. In consideration of this problem, the authentication mechanism is expected to be effective even in the presence of high packet loss.

In this paper, we propose a lightweight authentication protocol, which utilizes *one-way hash chain* to provide effective and efficient authentication for neighboring communications in MANETs. Our protocol is lightweight, scalable and tolerant of packet loss.

The rest of this paper is organized as follows: Section II gives a brief description on main related work; Section III describes our authentication protocol in details; a security analysis will be given in Section IV; we will evaluate the performance of the protocol in Section V; and Section VI concludes the paper.

II. Related Work

The idea of TESLA key is proposed in [1].TESLA uses one-way hashed chain to generate keys, and delays disclosure of keys to guarantee that a node receives the packet before another node can forge the packet with already released keys. But the security condition of TESLA requires clock synchronization, which is very difficult to achieve in mobile ad hoc networks, if not impossible.

The design of our protocol is motivated by LHAP (a Lightweight Hop-by-hop Authentication Protocol for Ad Hoc Networks) [2]. LHAP is a lightweight hop-by-hop authentication specially designed for ad hoc networks. It uses two keys: TRAFFIC key and TESLA key. TRAFFIC key is used to authenticate packets; and TESLA key is used to achieve trust maintenance by authenticating KEYUPDATE message. KEYUPDATE message is sent periodically to guarantee that the current released key is valid so that a malicious node will not be able to use an obsolete key to forge a packet. LHAP is not only a comprehensive authentication approach, by thoroughly describing key management and traffic authentication, but also proved to be computationally efficient. However, it requires two keys, which hence not only adds more complexity in authentication, but also needs to periodically send key maintenance packages that themselves need to be authenticated with TESLA keys. In addition, LHAP does not eliminate the disadvantage of delayed authentication in TESLA because the authenticity of the packets and the TRAFFIC key can not be verified until TESLA key is authenticated.

III. The Authentication Protocol

Our authentication protocol utilizes *one-way hash chains*, which is more efficient and less expensive than asymmetric cryptographic operations. One-way hash chain is a widely-used cryptographic primitive that uses a *one-way hash function* to generate a sequence of random values that serve as authentication keys. It has been used in authentication schemes for wireless ad hoc networks [3] and sensor networks [4].

Figure 1 demonstrates the one-way hash chain construction, utilization and revelation. To generate a key chain of length n+1, the first element of the chain h_0 is randomly picked and then the chain is generated by repeatedly applying a one-way function (denoted as H in Figure 1). A one-way hash function maps an input of any length to a fixed-length bit string, which is defined as $H:\{0,1\}^* \rightarrow \{0,1\}^\phi$, where ϕ is the length of the output of the hash function – the newly generated key. The function H should be simple to compute nonetheless must be computationally infeasible in general to invert. In utilization and revelation of these keys, we use the reverse direction of key generation: we start from h_n, the last generated, and then h_{n-1}, …, h_0. Any key of the one-way key chain commits to all previous keys[1], and h_n is a commitment to the entire one-way chain. Any key h_j can be verified from h_i $(0 \le i < j \le n)$ to be indeed an element in the chain by repeatedly applying H for $j-i$ times, that is, $h_j = H^{j-i}(h_i)$. Therefore, given an existing authenticated element of a one-way hash chain, it is possible to verify elements later in the sequence of use within the chain.

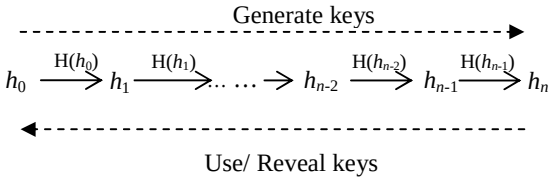


Figure 1: One way hash chain example

The chain of keys can be created all at once off-line before the mobile node joins the network and then stored for later use.

We use the following notations to describe our authentication protocol in this paper (see Table 1).

TABLE 1.
NOTATION OF THE PROTOCOL

Symbol	Description
A, *B*	Identities of mobile nodes
Cert_A	Certificate of node A's public key signed by CA's private key
$Sign_A(M)$	Digital signature of message M, signed with node A's private key
MAC(M, *K*)	MAC over message M with key *K*
h_i^A	The i[th] key in node A's one-way hash chain
H_A	Node A's hash function
H_A^k	Applying A's hash function for *k* times
M1|M2	The concatenation of message M1 and M2
P_i^A	The i[th] packet of node A's.

In this section, we will discuss the assumptions on which our protocol is established, which is followed by a detailed description on the basic scheme of our authentication protocol, including trust management and message authentication; and at last we will address the problem of key disclosure.

[1] In the sequel, when we refer to the direction of key generation as the direction of the chain. For example, the subsequent key of h_0 is h_1, and so on.

A. *Assumptions*

To prevent a malicious entity from forging packets with MACs that are computed using already released key, a packet sent by a node has to be received by an immediate neighboring node before a third party is able to replay the packet to it, unless the receiver has dropped the packet. This necessary condition for authentication using one hash key chain is assured in our approach by using delayed key disclosure. The scheme of key disclosure will be discussed later in this section.

We assume that each node can communicate with a trusted certificate authority (CA) before it enters the ad hoc network, and it can obtain a public key certificate signed by the CA as well as an authentic public key of the CA. The public key of the CA will be used to verify key certificates distributed by other nodes. However, a node may not be able to contact the CA after it joins the network because it is difficult for an ad hoc network to provide a central administration point since all the nodes in an ad hoc network are mobile. Moreover, a central entity is very likely to become the most vulnerable point in the network, which is subject to various malicious attacks.

We also assume that the mobile nodes that we are protecting are relatively underpowered so that asymmetric key operations such as digital signatures are too expensive for them to compute for each packet. In our scheme, digital signature is only used in trust bootstrapping so that the nodes can verify the genuineness of the first revealed key. Once the initial key is confirmed to be authentic, the subsequent keys can be verified by applying the one-way hash function.

On the contrary, the adversaries are powerful with the following capabilities: (1) an adversary can be capable of various attacks: eavesdrop, delay, drop, replay or alter packets; (2) an adversary's computation resources can be very large but yet limited. This means that an adversary may be able to conduct fast computations, such as computing MACs with negligible delay. The adversary, nevertheless, cannot invert a hash function and hence cannot obtain a hash key before the key owner reveals it.

B. *Trust Management*

1) Trust bootstrapping

To use one-way hash key chain for authentication, a node needs to distribute an authentic key such as h_n, which is the first revealed key from its generated chain. This key commits to the whole key chain and therefore the genuineness of the subsequent keys can be verified by applying hash function to this key, such as: given a key h_i, it is a genuine key from the chain if $h_n = H^{n-i}(h_i)$, or a counterfeit one otherwise.

Our scheme requires that a node contact the certificate authority to obtain public key of the CA as well as the certificate of the node's own public key before it joins an ad hoc network. The node can also pre-compute the whole one-way hash key chain off-line to reduce computational latency. Then the node signs the message with its private key and broadcasts a JOIN message to its neighbors. We suppose that a node, say node *A*, is sending JOIN message to its neighbors. The JOIN message will be in the following format:

$$A \rightarrow *: Cert_A, \{A | h_n^A | H_A\}, Sign_A(A, h_n^A, H_A)$$

where CertA denotes the certificate of node A's public key that has been signed by CA's private key; *A* denotes the identity of node A; and $Sign_A(A, h_n^A, H_A)$ denotes the digital signature of message $\{A | h_n^A | H_A\}$.

Upon receiving this JOIN message, every receiving node first uses CA's public key to verify the certificate of A's public key. Once the genuineness of node A's public key is confirmed, the key can be used to verify the digital signature on A's message. If the digital signature is validated to be authentic, the receiving node will record A's initial key h_n^A as well as its hash function H_A.

To bootstrap an authentic hash key to node A, each of its neighbors (say node *B*) unicasts the following ACK message to node A:

$$B \rightarrow A: Cert_B, \{B | h_m^B | H_B\}, Sign_B(B, h_m^B, H_B)$$

where h_m^B denotes B's most recently released key. Node A will perform the same verifications on B's ACK message as what node B did with A's JOIN message.

2) Trust maintenance

The trust relationship between a node and its neighbors is maintained with a periodical broadcast of KEYUPDATE message. In the KEYUPDATE message, a key that has been used to compute MACs will be released, and the neighboring nodes will verify the new released key with corresponding hash function. The maintenance process is described below:

Each node periodically broadcasts a KEYUPDATE message to its neighbors, which discloses its most recently used key:

$$A \rightarrow *: A, h_j^A$$

The key h_j^A will be authenticated by its neighbors based on the previously released key h_{j+1}^A: if it can be proved $H_A(h_j^A) = h_{j+1}^A$, the key h_j^A is considered valid; otherwise, the key is invalid and the receiving node may optionally issue an intrusion alert to other nodes.

3) Trust termination

In our authentication scheme, the trust relationship between two nodes may be terminated under two circumstances. First, when a node is detected to be compromised, the detecting nodes will permanently terminate their trust relationship with the compromised node. In this case, a further step such as excluding the node from the network might be taken. Second, when a node does not receive the KEYUPDATE message from a neighbor for a period that exceeds a predefined threshold, it will terminate its trust of the neighbor temporarily. This can happen when the neighboring node moves out of the node's transmission range, or when the neighboring node is not transmitting any data packets for a fairly long time (we assume that in case a node does not have any packets to send, it will not release key periodically in order to save its keys). If the two nodes want to restart their communications, they can run the trust bootstrapping process again to reestablish their trust relationship. The value of the threshold is

dependent on the size of the cache for authentication at the node. The cache is used to store the authentication information of other nodes', such as hash function, previously released key, and non-verified messages. A node with a larger cache can store more commitment information and therefore a trust relationship may be kept for longer time.

C. *Message Authentication*

When a node wants to send a message, it computes the MAC on the message and then unicast to the receiving node (say node B), or multicast (or broadcast) the packet (denoted as P^A) to the receivers in the following format:

$$A \rightarrow B(*) : M,\ MAC(M, h_i^A)$$

where h_i^A is the currently used key of node A's. Note that the key h_i^A has not been disclosed at this point. The originator of the packet (node A in this case) will later disclose h_i^A in KEYUPDATE message. The key enables the receiver to verify the MAC of the message. If the verification is successful, the message is then authenticated and trusted. Once the key is disclosed, it becomes obsolete and can not be used to generate MACs any more.

D. *Key Disclosure*

1) Security condition and threat model on authentication

This authentication protocol can be compromised if an adversary obtains node A's secret key h_i^A before a receiver receives the data packet that is protected with this key, because the adversary would be able to change the message and then use the key to re-compute the MAC of P^A, or even to forge all subsequent traffic. To prevent from this type of attacks, the receiver needs to be assured that it receives the data packet before the corresponding key is disclosed by the sender. The following *security condition* describes this requirement:

"A data packet *P* arrived safely, if the receiver receives the packet when the sender did not yet send out the corresponding key disclosure packet."

It is known that radio channels in MANETs are more prone to error than those in the Internet in that wireless communication links use open shared medium. The erroneous communication caused by signal conflicts may result in deteriorations of packets or even packet drops.

Figure 2 exemplifies an attack that takes advantage of KEYUPDATE packet drop to send maliciously modified or forged packets. Suppose node A is sending a message *Ms* to its neighbors with MAC (denoted by MAC(Ms, *K*) in Figure 2), which was generated with key *K*. Then A discloses key *K* to its neighbors B, C, D and M. Suppose node B does not immediately receive the message *Ms* and the KEYUPDATE message due to signal conflict at its channel. Node M, which is a malicious entity, then takes advantage of this chance to modify the message to *Ms*' and sends the tampered packet to node B with a MAC that is generated using the disclosed key *K* (denoted by MAC(Ms', *K*) in Figure 2). Node B would believe that it is a legitimate packet from A when it later receives the resent KEYUPDATE message from A (or a replayed KEYUPDATE message from node M).

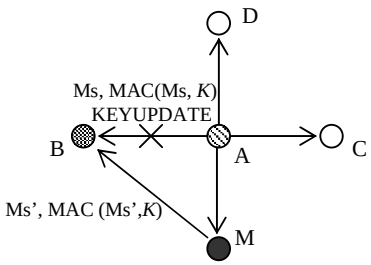


Figure 2: An example of in-the-middle attack on key disclosure

2) Delayed key disclosure

To prevent from the "in-the-middle" attacks described above, a receiver should have the knowledge of when to expect a KEYUPDATE message. TESLA uses delayed key disclosure to solve the problem. It also uses *time synchronization* to guarantee that the receiver can unambiguously verify if the security condition holds on each packet and then decide to keep or drop the packet. However, clock synchronization relies on two assumptions: first, the nodes to be synchronized have the ability to periodically exchange messages; and second, the nodes have the ability to estimate the time it takes for a message to travel between them. In mobile ad hoc networks, the high mobility of nodes lead to frequent reconfiguration of topology and frequent change of communication capacity between two nodes. Therefore, clock synchronization is very difficult (if not impossible) to achieve in a MANET in that there is no central control and packet delays may vary due to unpredictable mobility and radio interference.

Our authentication protocol uses delayed key disclosure without requirement for clock synchronization. In the protocol, a currently used key is broadcast after the key has been used to generate or verify MACs for a *time interval*. This time interval, namely *delay of key disclosure* in this context, is determined by the sender and announced in the data packets that are protected with the key. Before a key is disclosed, the packets with MACs that are computed with the key cannot be authenticated. Packets can be stored in cache at the receiving node until the key has been received and the authentication is completed.

We define the *delay of key disclosure*, denoted by *d*, as the time difference between key disclosure and the time when sender *starts* to send messages that use the key to compute MACs. Specifically, if a sender starts to send the first packet that is authenticated via MACs with key *K* at time t_0, then key *K* will be disclosed at time $t = t_0 + d$. Suppose there are *m* packets on which MACs are computed with key *K*: denoted by $P_1^K, P_2^K, \ldots, P_m^K$ respectively in sequence of being sent, and the times when they will be sent are $t_1^K, t_2^K, \ldots, t_m^K$ respectively. We denote the time interval between sending of the packet and the key disclosure as *r*, and the interval for packet *i* as r_i. The timeline is shown in Figure 3.

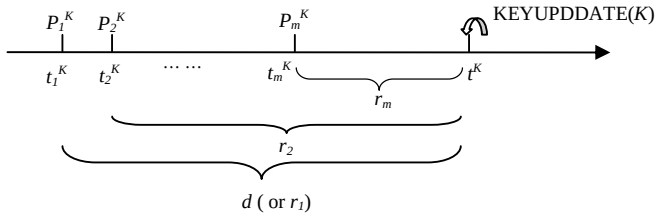


Figure 3: An example of the timeline for delayed key disclosure

In the example demonstrated in Figure 3, we have:

$$r_1 = t^K - t_1^K (= d);$$
$$r_2 = t^K - t_2^K;$$
$$\cdots\cdots$$
$$r_m = t^K - t_m^K.$$

In our protocol, a sender announces the remaining time *r* in its data packets[2]. The receiver can estimate when to expect the arrival of the KEYUPDATE message according to the remaining time. Suppose the receiver receives the packet $P_i^K (1 \le i \le m)$ at time rt_i^K. The remaining time indicated in the packet is r_i. In case that the data packet and the KEYUPDATE message are delivered at the same transmission rate, the KEYUPDATE message should arrive at the receiving end at time $rt^K = rt_i^K + r_i$. If the data packet and the KEYUPDATE message are delivered at different transmission rate (supposedly the difference is δ), then the KEYUPDATE message should be expected at the receiving end at $rt^K = rt_i^K + r_i + \delta$. δ can be estimated at each node according to its observation on the traffic .

This scheme eliminates the need for clock synchronization, which is used in TESLA. Although it still needs to estimate the difference between transmission rates of a data packet and its KEYUPDATE message, it is easier than clock synchronization because it does not need to estimate the absolute value of transmission delay. Instead, it only needs to estimate the variance of the transmission delays on data packets and the corresponding KEYUPDATE message, which is much easier.

In our protocol, it is possible that a key (say h_i) is disclosed after the packets using the next key h_{i-1} have been sent. Therefore, the receiver needs to know which key is used for which packets. To solve this problem, we include the index of the key in data packets, so that the receiver will be able to know which key should be used to authenticate the message. Therefore, a data packet from node A destined to all its neighbors (broadcast) or to node B (unicast) is in the following format:

$$A \rightarrow *(B) : M, MAC(M), r, index$$

[2] Note that the time when a packet will be sent can not be exactly known at the time of packet generation. However, it can still be accurately predicted according to the cache status at each node.

where *index* denotes the index of the key that will be used to authenticate the message. And the KEYUPDATE message will be:

$$A \rightarrow *: A, \quad h_j^A, index$$

The index of the key is not protected in the message. In case that it is tampered such as maliciously increased or decreased, it can still be verified by repeatedly applying hash functions to the key until the result matches the previously received key and meanwhile counting how many times the function has been applied. For example, if the newly arrived key is K and the previously received key is K' and $K' = H^n(K)$, then $index(K) = index(K') + n.$

Using this method, our protocol is tolerant of packet loss because the key verification is not based on the immediate previous key.

In our scheme, the delay of key disclosure can vary for different keys. It is not a predetermined and unchanging value since establishment of the trust relationship, as what TESLA has used. The advantage of varying delays of key disclosure is that it allows a sender to choose key disclosure period according to the pattern of the traffic transmitted by the sender: when the traffic is heavier, the delay should be smaller; and vice versa. This can prevent the cache from being "flooded". An example of this varied delays scheme is demonstrated in Figure 4.

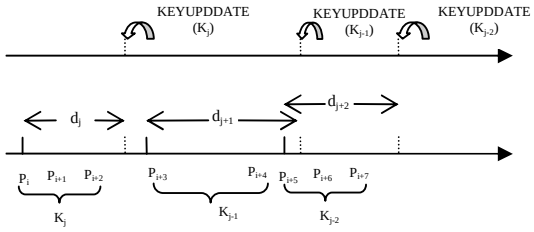


Figure 4: Varied delays of key disclosure

3) Comparison with TESLA key disclosure scheme

The differences between our key disclosure and that of TESLA are:

- We broadcast KEYUPDATE message to release keys, while TESLA releases keys in data packets. Because different data packets may be targeted at different groups of receivers, TESLA is not able to guarantee that the key would be disclosed to all the receivers that have received the data packets protected by the key.

- Our protocol eliminates the need for clock synchronization. Clock synchronization has been proved to be prohibitively difficult and therefore we argue that it should be used in authentication mechanisms.

- In our protocol, the delay of key disclosure is not a fixed value since configuration of the network, as TESLA has used. It is up to the sender to decide the delay values based on the traffic status of the network. It allows more flexibility than TESLA and avoids the problem of authentication cache overflow.

IV. Security Analysis

In this section, we will analyze the security properties of our protocol and compare with digital signatures and the protocol of LHAP, which is also a lightweight authentication protocol designed specifically for ad hoc networks.

1) Trust management

Our protocol uses digital signature in both initial trust establishment and subsequent trust reestablishment. Compared to the scheme that uses asymmetric cryptography in only initial trust bootstrapping, our protocol can guarantee the genuineness of the key that commits to all the subsequent keys, and an "in-the-middle" attacker would not be able to replay an already released key and forge packets with the obsolete key afterward.

2) Message authentication

Up to date, MD5 [5] and SHA-1[17] are two of the most widely used cryptographic hash functions. MD5 has been recently shown to be vulnerable to collision search attacks [7]. This type of attacks and other currently known weaknesses of MD5 can be thwarted by the use of MD5 within HMAC [8]. MD5-HMAC is proved to be more secure than MD5 in protecting the authenticity of traffic.
Our message authentication can effectively thwart the attacks of forging or maliciously alteration of packets.

3) Key disclosure

The delayed key disclosure can prevent from in-the-middle attack in which an adversary may use an obsolete key to forge or alter packets. However, the performance is dependent on the value of the delay.

Non-repudiation is also achievable in case of using large delay values.

V. Simulation and Performance Analysis

In this section, we will evaluate our trust management and message authentication as well as the delayed key disclosure approach.

A. Simulation Setup

We use Network Simulator, ns2 [10], for our simulations. The routing protocol we used in our simulation is AODV. The Medium Access Control (MAC) protocol is IEEE 802.11 and the Transportation layer protocol is User Datagram Protocol (UDP), which are both available as a part of the simulator. The size of data packets is 512 bytes the traffic sources are Constant-Bit-Rate (CBR). We assume all the nodes have the same initial transmission range of 250 meters.
In our simulation, all traffic is generated and the statistical data are collected after a warm-up time of 100 seconds in order to allow the network to finish initialization process.

Scenario 1: The first scenario we used is demonstrated in Figure 5. There are totally nine nodes in the scenario. Eight of them (denoted as N1, N2, … , N8 in the figure) serve as transmission

nodes, who transmit packets to one single receiving node (denoted as N9 in the figure). Node N9 is the sink of all the traffic. The nodes are positioned at the mesh that is demonstrated in the figure. Static network topology used in this scenario allows us to easily observe the network performance (such as hop-by-hop delay, etc) according to varied channel loads.

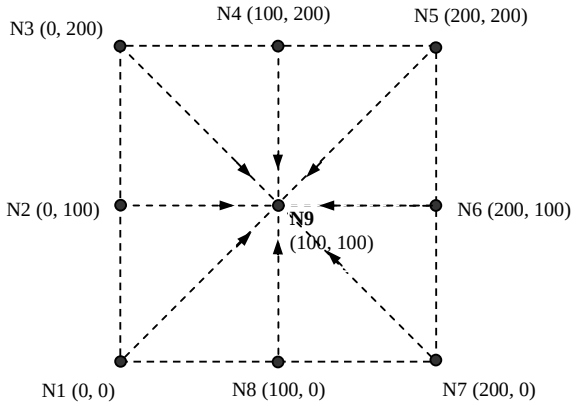


Figure 5: Network topology of 9-node scenario

Scenario 2: In our second scenario, 50 mobile nodes are randomly distributed in a 1500x300 rectangular space. The node mobility model is random waypoint model, which is commonly used in simulations for mobile ad hoc networks. The maximum node speed is 20 m/s.

B. Performance Evaluation on Trust Management and Message Authentication

The performance metrics employed to analyze our system are: *computational overhead*, *authentication latency*, *message overhead*.

1) Computational overhead

As any authentication mechanisms, our protocol introduces computational overhead by two operations: message authentication and trust management.

In our protocol, symmetric cryptography is used for message authentication. It is known that symmetric cryptographic operations are three to four orders of magnitude faster than asymmetric operations, especially on CPU limited devices.

We used asymmetric cryptography in trust bootstrapping, that is, when a node is establishing or reestablishing trust relationship with its neighboring nodes. This may introduce more overhead than LHAP because LHAP employs digital signature only when the trust is bootstrapped for the first time. However, we have argued that using digital signature is necessary even in re-bootstrapping since the key release is vulnerable to replay attack, especially when the receiving node has moved out of transmission range for a time interval hence is likely to be unaware of the currently released key. It will not introduce significant overhead on receivers because signature verification is much faster than signature generation [9].

Moreover, our protocol only maintains one authentication key, which consumes much less resource such as CPU and memory than LHAP, which maintains two keys – TRAFFIC key

and TESLA key. We only use digital signature for trust bootstrapping. The trust maintenance is still based on one-way hash function, which is so efficient that is usually considered negligible.

2) Authentication latency

The latency of authenticating a packet is introduced by two parts: MAC verification latency and key disclosure delay.

MAC verification is accomplished by computing one hash. The latency for this verification is less than one millisecond even for very constrained computational capability such as handheld PDAs [9]. Therefore, the authentication latency is mainly determined on the key disclosure delay.

The delay of key disclosure is a value that is determined by the sender of packets based on the traffic pattern. A very small delay may cause difficulty in satisfying the security condition and consequently increase the risk to key replay attack; while large delay may result in an increase on authentication latency. Tradeoff should be made between performance and security properties. A quantity analysis on the delay of key disclosure is included later in this section.

3) Message overhead

Message overhead is introduced by trust management messages (such as trust bootstrapping, KEYUPDATE and trust relationship termination messages) and MACs of packets.

Suppose that the authentication is performed using MD5 Message Digest Algorithm. Then the MAC attached to each packet is a hashed digest that is 128-bit long. If the data packet size is 512 bytes, the overhead introduced by MACs is approximately 3%, which is very small.

The overhead introduced by trust management varies with the frequency of bootstrapping and KEYUPDATE messages. It is obvious that high node mobility will result in more frequent trust bootstrapping and therefore introduce more overhead. In addition, a node sending more traffic will lead to more frequent broadcast of KEYUPDATE messages, which also introduces more overhead.

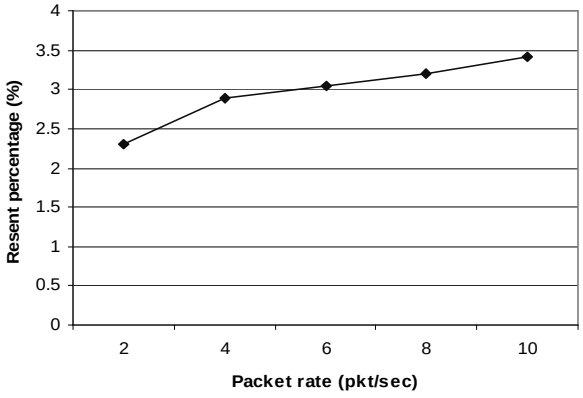


Figure 6: Average resent rate of KEYUPDATE messages

Figure 6 demonstrates the simulation results of the KEYUPDATE messages that have been resent. Data packet rates that are used in our simulation vary from 2 pkt/sec (packets per second) to 10 pkt/sec. The KEYUPDATE messages are sent with the same rate of the data packets. This implies that we use a new key for each data packet, which is the worst case for key update scheme in term of message overhead.

We can tell from Figure 6 that the resent rate of KEYUPDATE packets increases with the increase of the data packet rate. We assume that the identification of a node is128-bit long and the index of authentication key is 128-bit long too. In case that the data packet rate is 10 pkt/sec, when the resent rate is the highest in these scenarios, the message overhead introduced by KEYUPDATE messages is only 9.7%. If we use the less frequent KEYUPDATE messages (such as one per 3 seconds), the message overhead is negligible.

C. *Performance Analysis on Delayed Key Disclosure*

To analyze our delayed key disclosure scheme, we first take a measurement on average hop-by-hop delay. H*op-by-hop delay* of data packets is an important metric in determining the value of the delay that should be used in key disclosure scheme, in that the key disclosure delay should be large enough to guarantee arrival of the data packets before the key but meanwhile be as small as possible to achieve low authentication latency. We use hop-by-hop delay instead of end-to-end delay because our authentication protocol is designed for neighboring communications and the transmissions the protocol is aimed to protect are only one-hop transmissions.

Then we will use different key disclosure delay values to evaluate the performance, in metrics such as *percentage of packets arriving safely* and *dropped packet rate*.

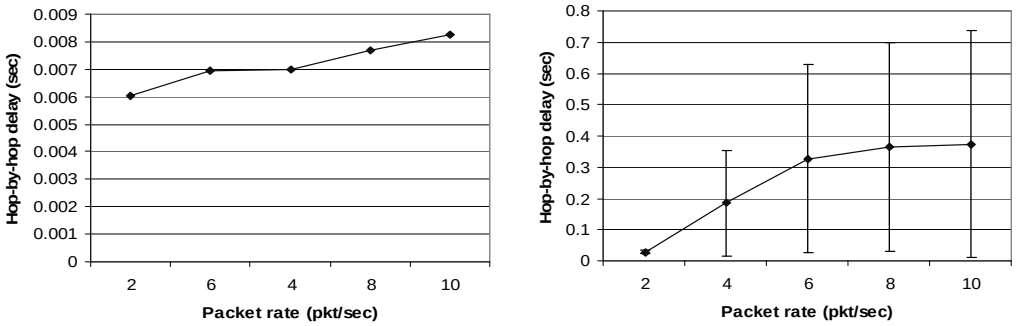
1) Average hop-by-hop delay

We measured average hop-by-hop delay on both Physical Layer level and Network Layer. The delay on the Physical level is mostly the transmission time the packet takes in the air. We tested it in the scenario where there are two nodes, one of which transmits packets to the other. The distance between the two nodes is 150 meters. The average delay is 0.00467269 second with a standard deviation of less than $1x\ 10^{-6}$ second.

The average hop-by-hop delay at the network layer is tested in both the scenarios of 9 nodes and 50 nodes we described earlier in this section. The hop-by-hop delay is calculated as *end-to-end delay* (a packet takes from the source to the destination) divided by the number of links a packet has traversed during delivery from the source to destination (the number of hops), i.e.

$$hop-by-hop\ delay = \frac{end-to-end\ delay}{number\ of\ hops}$$

We measured the delay at the Network Layer because the key disclosure delay value (denoted as *r* in previous sections) will be determined and stamped on data packets above the Medium Access Control Layer level. Above Medium Access Control Layer, data packet delay may result not only from the transmission in the air but also from the *backoff* due to channel contention at Medium Access Control layer and from the *queue delay*.



(a) Hop-by-hop delay in Scenario 1 (b) Hop-by-hop delay in Scenario 2

Figure 7: Average Hop-by-hop Delay

The results for Scenario 1 (9 nodes) are shown in Figure 7(a). The deviations are too small (less than 0.00002 second for all the cases) to be shown in the figure.

We tested average hop-by-hop delay in Scenario 2 with varied pause time, which is changed from 60 seconds to 480 seconds in an interval of 60 seconds (see Figure 7 (b)). The hop-by-hop delay for each packet rate is the average value from the cases with different pause time. The vertical line at each point represents the *standard deviation* of the value.

We can see from Figure 7 that average hop-by-hop delay increases with the increase of the data packet rate. The reason for this increasing delay is that increased packet rates result in larger channel load and therefore more channel contention for packets, and the channel contention causes more backoff time for data packets. Table 2 and Table 3 give the average channel loads[3] according to the packet rates in Scenario 1 and 2 respectively.

TABLE 2
AVERAGE CHANNEL LOADS (SCENARIO 1: 9 NODES)

Packet rate (pkt/sec)	Channel load (bps)	Channel load Percentage (%)
2	173974	8.70
4	374926	18.75
6	534254	26.71
8	733054	36.65
10	907016	45.35

TABLE 3
AVERAGE CHANNEL LOADS (SCENARIO 2: 50 NODES)

Packet rate (pkt/sec)	Channel load (bps)	Channel load Percentage (%)
2	57096	2.85
4	107592	5.38
6	125818	6.29
8	156477	7.82
10	182633	9.13

With the same data packet rate, the average channel loads in Scenario 2 are less than the corresponding channel loads in Scenario 1. However, the delay values are larger than those in the scenario of 9 nodes. This is caused by the following reasons:

First, channel loads do not always accurately reflect the contention status at a channel, because a node's neighboring communications may also affect its capability of receiving packets and

[3] Please note that here "channel" refers to the medium that a nodes shares with all its neighbors, which is different from "link", which refers to the point-to-point medium that two neighboring nodes use for transmission.

the packets in these communications are not accounted as its channel load. In Scenario 2, although the channel loads are lighter, the contention is more intensive in that most of the nodes have more neighbors than node N9 in the first scenario. As we have mentioned earlier, more intensive contentions result in more backoffs and hence larger transmission delays.

Second, node mobility may also introduce delays since it can cause re-routing when the network topology changes. These routing packets will compete with data packets for the bandwidth of channels and therefore cause more backoffs on data packets.
From the above simulations, we can conclude that *hop-by-hop delay increases with increase of traffic load in the neighborhood*. Therefore, a sender should use larger key disclosure delay in case of heavier traffic load.

2) Percentage of packets arriving safely

According to the average hop-by-hop delay demonstrated in Figure 7, we tested our key disclosure scheme with varied disclosure delay values. The percentages of data packets that arrive safely according to different data packet rates are shown in Figure 8. We observe that more than 97.6% of the data packets have arrived safely when the key disclosure delay is set to 3 seconds; more than 94.8% of the data packets have arrived safely if the key disclosure delay is set to 2 seconds, in all the cases of different data packet rates.

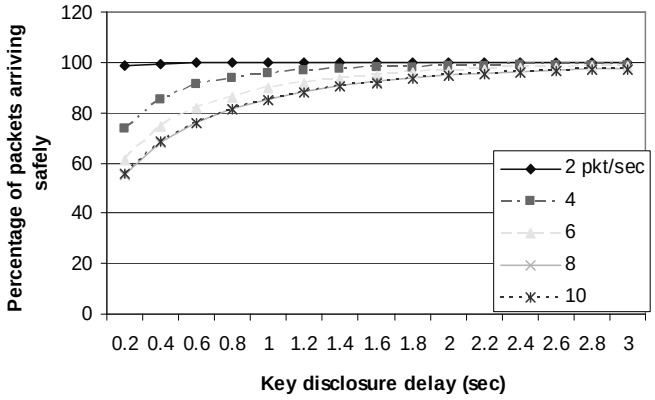


Figure 8: Percentage of packets arriving safely

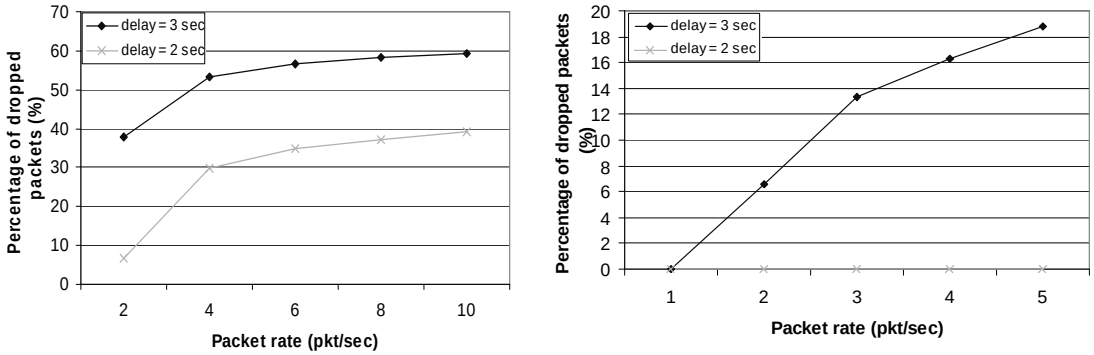
3) Dropped packet rate

We also test the dropped packet rates with different cache sizes. We use very small cache sizes (16 packets and 32 packets) to observe the performance. The key disclosure scheme should have less dropped packet rates in real networks since larger cache sizes (such as 128 packets) are often used.

The results for the two cache sizes are shown in Figure 9 (a) and Figure 9 (b) respectively. From the simulation, we have noticed that with increase of data packet rates, the drop rate at the cache increases too. We can also observe that, if the cache size is as small as 16 packets, there will be about 39% data packets dropped at the cache at 10 packets per second of data packet rate if the key disclosure delay is set to 2 seconds. In case of 3 seconds key disclosure delay, the drop rate will increase to 60% or so. With the cache size of 32 packets, drop rate

decreases to 0 in case of 2 seconds or lower key disclosure delay in case that the data packet rate is 10 packets per second. If the key disclosure delay is 3 seconds, the drop rate is about 19%. However, if we use a cache with size of 64 packets, the drop rate will drop to 0 no matter what the data packet rate is (in a 2 pkt/s to 10 pkt/s range).

If we use a cache of length 64-byte, the dropped packet rate will be 0 even with 10 pkt/sec data packet rate.



(a) Dropped packet rate (cache size: 16 packets) (b) Dropped packet rate (cache size: 32 packets)

Figure 9: Average dropped packet rate

VI. Conclusion

Most ad hoc networks do not employ any network access control, leaving them vulnerable to resource consumption attacks. In ad hoc networks, users need to assure the party who supposedly sent a message to another party is indeed the legitimate party. Otherwise, a malicious node could tamper a network with falsified data. These attacks can result in degraded performance of networks, interference of resource reservation, and unauthorized use of resources. To deal with these attacks, an authentication protocol needs to be in place to ensure that a packet is sent by an authentic and legitimate node.

In this paper, we have proposed a lightweight authentication protocol that effectively and efficiently provides security properties such as authenticity and integrity for communicating neighbor nodes in MANETs. The protocol utilizes *one-way hash chains* to compute authentication keys, which not only eliminates the high performance overhead imposed by *asymmetric* cryptography (such as digital signatures), but also avoids the difficulty of key management introduced by secret paired *symmetric* key. Our protocol also used *delayed key disclosure* to prevent a malicious entity from forging packets with Message Authentication Codes (MACs) with an already released key.

The authentication protocol is lightweight, scalable and tolerant of packet loss. The performance analysis showed that the protocol incurs low overhead penalty and also achieves a tradeoff between security and performance. The delayed key disclosure approach can achieve an extremely low dropped packet rate if the data packets are cached in a fair size buffer before being authenticated.

References

[1] A. Perrig, R. Canetti, J. Tygar, D. Song. "Efficient authentication and signing of multicast streams over lossy channels". In *Proc. of IEEE Symposium on Security and Privacy*. May 2000.

[2] S. Zhu, S. Xu, S. Setia, and S. Jajodia. "LHAP: A Lightweight Hop-by-Hop Authentication Protocol for Ad-Hoc Networks". In *ICDCS 2003 International Workshop on Mobile and Wireless Network (MWN 2003)*, Providence, Rodhe Island, May 2003.

[3] Y. Hu, D. Johnson, A. Perrig. "SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks". In *Proc. of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002), IEEE*, Calicoon, NY, June 2002.

[4] A. Perrig, R. Szewczyk, V. Wen, D. Culler, J. D. Tygar, "SPINS: security protocols for sensor networks", In *Proceedings of the 7th annual international conference on Mobile computing and networking*, July 2001.

[5] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[6] NIST, FIPS PUB 180-1: "Secure Hash Standard", April 1995.

[7] H. Dobbertin, "The Status of MD5 After a Recent Attack", RSA Labs' CryptoBytes, Vol. 2 No. 2, Summer 1996.

[8] H. Krawczyk, M. Bellare and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[9] M. Brown, D. Cheung, D. Hankerson, J. Hernandez, M. Kirkup, and A. Menezes. "PGP in Constrained Wireless Devices". In *9th USENIX Security Symposium*. August 2000.

[10] The network simulator - ns-2, http://www.isi.edu/nsnam/ns/.

Bin Lu is a Ph.D. candidate in the Department of Computer Science at Texas A&M University. She obtained her B.S. (1996) and M.S. (1998) degrees in computer science from Harbin Institute of Technology, China. She will be joining West Chester University in Pennsylvania as an Assistant Professor in Computer Science in August 2005. Her research interests are Network Security, Quality of Service and Mobile Ad Hoc Networks.

Udo W. Pooch, P.E., is the E-Systems Professor of Computer Science at Texas A&M University. He received his Ph.D. in theoretical physics from the University of Notre Dame. Dr. Pooch has authored numerous articles and books. His research interests include network simulation, network security, and fault-tolerant distributed environments.