Available online at www.sciencedirect.com
ScienceDirect
Ad Hoc Networks 5 (2007) 1194–1203
Ad Hoc Networks
www.elsevier.com/locate/adhoc

Non-interactive key establishment in mobile ad hoc networks ☆

Zhenjiang Li [a,*], J.J. Garcia-Luna-Aceves [a,b]

[a] Department of Computer Engineering, University of California, Santa Cruz, 1156 High Street, Santa Cruz, CA 95064, United States
[b] Palo Alto Research Center (PARC), 3333 Coyote Hill Road, Palo Alto, CA 94304, United States

Received 15 July 2006; accepted 17 July 2006
Available online 22 August 2006

Abstract

We present a new non-interactive key agreement and progression (NIKAP) scheme for mobile ad hoc networks (MANETs), which does not require an on-line centralized authority, can non-interactively establish and update pairwise keys between nodes, is configurable to operate synchronously or asynchronously, and supports differentiated security services w.r.t. the given security policies. NIKAP is valuable to scenarios where pairwise keys are desired to be established without explicit negotiation over insecure channels, and also need to be updated frequently.

Keywords: Self-certified key (SCK); NIKAP; Ad hoc on-demand routing

1. Introduction

There are three cryptographic techniques commonly used in practice to devise security mechanisms for MANETs: one-way hash functions, symmetric cryptosystems and asymmetric (or public key) cryptosystems. An asymmetric cryptosystem is more efficient in key utilization in that the public key of a node can be used by all the other nodes, while a symmetric cryptosystem requires the existence of a shared key between two communicating nodes. Hash functions can be implemented quickly, and usually work together with symmetric or asymmetric algorithms to create more useful credentials, such as a digital certificate or a keyed hash value.

Portable devices forming a MANET usually have limited battery life-time and must share a relatively limited transmission bandwidth. Therefore, symmetric cryptosystems are preferable in ad hoc scenarios due to their computational efficiency (conducting an asymmetric algorithm usually is three or four orders of magnitude slower than the symmetric counterpart). The key establishment problem between two network principals is well understood for conventional communication networks, and generally can be resolved by key distribution or key agreement.

The classic key-distribution scheme, such as Kerberos [1], requires an on-line centralized authority (CA) to generate and distribute keys for nodes. However, this is not suitable for MANETs. In practice, the on-line CA can be unavailable to some of the

☆ This work was supported in part by the Baskin Chair of Computer Engineering at UCSC, the National Science Foundation under Grant CNS-0435522, and the US Army Research Office under Grant No. W911NF-05-1-0246.
* Corresponding author. Tel.: +1 831 4595436; fax: +1 831 4594829.
E-mail addresses: zhjli@soe.ucsc.edu (Z. Li), jj@soe.ucsc.edu (J.J. Garcia-Luna-Aceves).

1570-8705/$ - see front matter
doi:10.1016/j.adhoc.2006.07.002

nodes, or even the whole network during certain time periods, because of the unpredictable state of wireless links and node mobility. Given that the CA is the single point of failure, compromising the CA jeopardizes the security of the entire system. More importantly, the Kerberos system is designed to provide authentication and key distribution services for networks structured according to the client-service model, while a MANET is a peer-to-peer communication system for the purpose of routing. Recently proposed key distribution protocols [2] replace the functionality of CA by a subset of nodes in the network. However, given that applications need to contact multiple nodes that can be multiple hops away, to obtain the desired keys, it is not clear whether sharing the CA functionality amongst multiple nodes can perform better than using a single CA.

Key agreement protocols, such as the Diffie–Hellman key exchange protocol [3] and many variations derived from it, do not need an on-line CA and compute the shared keys between nodes on-demand. These protocols are interactive schemes and active routes must pre-exist for such approaches to work. However, assuming that routes pre-exist contradicts the need to secure the routing discovery amongst nodes in the first place. Network dynamics can also tear them down in middle of the key negotiation, and as such no key can be agreed upon. Moreover, interactive schemes are not scalable because messages exchanged for key establishment consume significant CPU cycles and wireless bandwidth.

Motivated by the observations above and based on self-certified key (SCK) [4] cryptosystem, we propose new non-interactive key agreement and progression (NIKAP) protocols to facilitate the key agreement process in MANETs. NIKAP needs the aid of a centralized authority (CA) only at the initial network formation, and the CA can be entirely off-line thereafter. Scarce battery and bandwidth of wireless nodes are also saved in transmitting, receiving and processing messages. Though there are protocols establishing keys between nodes non-interactively (not rekeying) [11], to our knowledge, NIKAP is the first scheme that supports non-interactive key agreement and subsequent key progression simultaneously.

Section 2 reviews the basics of self-certified key (SCK) cryptosystem, which was first introduced by Petersen and Horster [4]. Section 3 presents S-NIKAP and A-NIKAP, the non-interactive key agreement and progression protocols tailored for MANETs. Sections 4–6 present the results of our recent use of NIKAP to secure the on-demand ad hoc routing, in which we show that NIKAP bootstraps key establishment in MANETs efficiently.

2. Overview of SCK

Self-certified key (SCK) system adopts the ideal of implicit verification, in which the authenticity of a public key is not verified until it is used for some cryptographic operations (e.g., signature verification and key exchanging). In the following, we first summarize the basic primitives used in SCK.

2.1. Initialization

A centralized authority (CA) Z is assumed to exist before the network formation. Z chooses large primes p, q with $q|(p-1)$ (i.e., q is a prime factor of $p-1$), a random number $k_A \in Z_q^*$, where Z_q^* is a multiplicative subgroup with order q and generator α; then Z generates its (public, private) key pair (x_Z, y_Z). We assume that the public key y_Z is known to every node that participates in the network. To issue the private key for node A with identifier ID_A, Z computes the signature parameter $r_A = \alpha^{k_A} (\mathrm{mod}\, p)$ and $s_A = x_Z \cdot h(ID_A, r_A) + k_A (\mathrm{mod}\, q)$, where $h(\cdot)$ is a collision-free one-way hash function and $(\mathrm{mod}\, p)$ means modulo p. Node A publishes the parameter r_A, called the guarantee, together with its identifier ID_A, and keeps $x_A = s_A$ as its private key. The public key of A can be computed by any node that has y_Z, ID_A and r_A using the following equation:

$$y_A = y_Z^{h(ID_A, r_A)} \cdot r_A (\mathrm{mod}\, p). \tag{1}$$

We denote this initial key pair as $(x_{A,0}, y_{A,0})$.

2.2. User-controlled key pair progression

Node A can update its (public, private) key pair *either synchronously or asynchronously*. In the synchronous setting, where A uses the key pair $(x_{A,t}, y_{A,t})$ in time interval $[t \cdot \Delta T, (t+1) \cdot \Delta T)$, node A can choose n random pairs $\{k_{A,t} \in Z_q^*, r_{A,t} = \alpha^{k_{A,t}} (\mathrm{mod}\, p)\}$, where $1 \leqslant t \leqslant n$, and publishes guarantees $r_{A,t}$. Then the private key of node A progresses as follows:

$$x_{A,t} = x_{A,0} \cdot h(ID_A, r_{A,t}) + k_{A,t} (\mathrm{mod}\, q) \tag{2}$$

and the corresponding public keys can be computed according to

$$y_{A,t} = y_{A,0}^{h(ID_A, r_{A,t})} \cdot r_{A,t} (\mathrm{mod}\, p). \tag{3}$$

2.3. Non-interactive pairwise key agreement and progression

Pairwise shared keys between any two nodes A and B can also be computed and updated synchronously or asynchronously as follows:

Node A:

$$x_{A,t} = x_{A,0} \cdot h(ID_A, r_{A,t}) + k_{A,t},$$
$$y_{B,t} = y_{B,0}^{h(ID_B, r_{B,t})} \cdot r_{B,t} (\mathrm{mod}\, p),$$
$$K_{A,t} = y_{B,t}^{x_{A,t}} (\mathrm{mod}\, p),$$
$$K_t = h(K_{A,t}).$$

Node B:

$$x_{B,t} = x_{B,0} \cdot h(ID_B, r_{B,t}) + k_{B,t},$$
$$y_{A,t} = y_{A,0}^{h(ID_A, r_{A,t})} \cdot r_{A,t} (\mathrm{mod}\, p),$$
$$K_{B,t} = y_{A,t}^{x_{B,t}} (\mathrm{mod}\, p),$$
$$K_t = h(K_{B,t}).$$

The pairwise shared keys obtained by node A and node B are equal because

$$h(K_{A,t}) = h\left(y_{B,t}^{x_{A,t}} (\mathrm{mod}\, p)\right) = h(\alpha^{x_{A,t} x_{B,t}} (\mathrm{mod}\, p)) = h\left(y_{A,t}^{x_{B,t}} (\mathrm{mod}\, p)\right) = h(K_{B,t}). \quad (4)$$

Unlike a certificate-based approach, guarantees of nodes can be published and need not to be certified (signed) by any centralized authority. This means that the public key of each node can be derived and updated (rekeying) without the aid of an on-line CA. Moreover, any pair of nodes can establish and progress pairwise keys between them in a non-interactive manner. Hence, without considering the distribution of guarantees, the overhead incurred by key establishment is zero.

3. S-NIKAP and A-NIKAP

For NIKAP to work correctly, we assume that the guarantees of a node are successfully distributed to all nodes participating in the network. To ensure the delivery of nodal guarantees in such error-prone environments as MANETs, an efficient and reliable broadcasting scheme, for instance the reliable broadcasting protocol proposed in [12], can be used to facilitate the process of guarantee distribution, which tolerates link failures and node mobility. Algorithms 1 and 2 present the specifications of S-NIKAP and A-NIKAP, respectively.

Algorithm 1. Protocol S-NIKAP

1: **Node (A) initialization:**
Retrieve CA's public key y_Z, initial private key $x_{A,0}$, initial guarantee $r_{A,0}$ and key progression interval ΔT
2: **Guarantees distribution:**
Advertise ID_A and randomly selected guarantees $r_{A,t}$ where $1 \leqslant t \leqslant n$ ($r_{A,t}$ and ID_A can be broadcast over insecure channel)
3: **Pairwise keys agreement and progression:**
To communicate with node B within time interval $[T_0 + t \cdot \Delta T, T_0 + (t+1) \cdot \Delta T)$, first update the key shared with B to K_t, according to the following procedure:
$x_{A,t} = x_{A,0} \cdot h(ID_A, r_{A,t}) + k_{A,t}$
$y_{B,t} = y_{B,0}^{h(ID_B, r_{B,t})} \cdot r_{B,t} (\mathrm{mod}\, p)$
$K_{A,t} = y_{B,t}^{x_{A,t}} (\mathrm{mod}\, p)$
$K_t = h(K_{A,t})$

Algorithm 2. Protocol A-NIKAP

1: **Node (A) initialization:**
Retrieve CA's public key y_Z, initial private key $x_{A,0}$ and initial guarantee $r_{A,0}$
2: **Guarantees distribution:**
Advertise ID_A and randomly selected guarantees $r_{A,t}$ where $1 \leqslant t \leqslant n$ ($r_{A,t}$ and ID_A can be broadcast over insecure channel)
3: **Random bits stream generation and exchange:**
To communicate with node B, first generate a random bits stream $BITS_A$ and send to B as follows
$A \Rightarrow B : \{ID_A, ID_B, BITS_A, \mathbf{hash}(ID_A, ID_B, BITS_A, K_{A,0})\}_{K_{A,0}}$
where hashing value $\mathbf{hash}(\cdot)$ is used by node B to verify the integrity of $BITS_A$
4: **Bit-Controlled key progression:**
while $BITS_A$ is not empty **do**
  If new session **then** ▷ *Or other triggering events*
    flag ← *pop*($BITS_A$)
    If flag = 1 **then**
      update to K_t
    else
      keep using K_{t-1}
    end if
  end if
end while

In synchronized NIKAP (S-NIKAP), where time synchronization is made available to each node, two nodes negotiate and update the shared keys between them periodically according to the current time instant and the specified security policy. Processes or applications of higher security concern can perform the rekeying operation at a high rate, and those of lower security concern at a low rate, accordingly. Therefore, communication principals in the network can be distinguished based on different security policies, such as roles, service types, or the sensitivity of data. As a result, differentiated security services can be achieved by specifying high-to-low rekeying rates that correspond to high-to-low security levels. The main limitations of S-NIKAP are the prerequisite of time synchronization and the periodical rekeying. Though there exist devices or protocols providing time synchronization for MANETs, it is still not clear if the desired performance can be achieved in such a dynamic and unpredictable environment. Moreover, a pairwise key is independently updated no matter whether there is communication between peer nodes to take place. Therefore, local CPU cycles and battery are wasted if the newly generated keys are not used within its life-cycle.

It follows naturally that an asynchronous version of NIKAP is desired in cases in which time synchronization is not available or portable nodes cannot afford the cost of progressing keys at high rates. Asynchronous NIKAP (A-NIKAP) has the same non-interactive rekeying capability as S-NIKAP does, but requires no time synchronization service from the underlying network. Instead, A-NIKAP uses a pseudo-random bit stream to *synchronize* the rekeying process between nodes, of which "1" invokes new key progression while "0" keeps two nodes using the current key shared between them. According to SCK, an initial shared key can be non-interactively established. Therefore, the pseudo-random bits stream can be generated, encrypted (using the initial key), and securely agreed upon between nodes sharing the initial key. If the same pseudo-random number generator is used by both ends, to save the bandwidth, only a common seed needs to be exchanged. The progression strategy in A-NIKAP can be specified as per-session based, fixed number of sessions based or fixed number of packets sent based etc., according to the given security policies. If we count one bit in the random bits stream equal to one time interval used in S-NIKAP, A-NIKAP incurs half of the local CPU cycles than S-NIKAP does, provided that the bits stream is perfectly randomized.

4. Ad hoc on-demand secure routing (AOSR) protocol

AOSR is an ad hoc on-demand secure routing protocol that derives pairwise keys using NIKAP, and exploits k-MAC to authenticate the generic on-demand ad hoc routing. We assume that each pair of nodes in the network shares a pairwise key $K_{i,j}$, which can be achieved by using the key agreement protocols described in Section 3. We also assume that the MAC (media access control) address of a node cannot be changed once it joins the network. Even though some vendors of modern wireless cards do allow a user to change the card's MAC address, we will see that this simple assumption can be helpful in detecting some complicated attacks such as wormhole. Moreover, every node must obtain a certificate signed by the CA, which binds its MAC and *ID* (can be the IP address of this node), before it joins the network. Note that such certificates are used for nodes to verify the authenticity of their neighbors, rather than validating the routes discovered during the process of route discovery. A node presents its certificate to each node that it meets for the first time, and two nodes can communicate with its neighbor nodes only if their certificates have been mutually verified. The approach used to authenticate and maintain neighbor-node information is presented in [5], and as such is omitted here due to the space limitations. In our discussion, *k-MAC* refers to keyed-message authentication code (a keyed hash value), while *MAC* refers to media access control unless specified otherwise. To be clear, the notation we use is summarized in Table 1.

AOSR consists of *route request initialization, route request forwarding, route request checking at destination* and the symmetric *route reply initialization, route reply forwarding and route reply checking at source*.

Route request initialization: Source S generates the following route request $RREQ$ and broadcasts to its neighboring nodes, when S wants to communicate with node D but has no active route maintained for D at that point

$$RREQ = \{RREQ, S, D, QNum, HC, \{NodeList\}, QMAC_{s,d}\}. \quad (5)$$

Table 1
Notation used in the paper

Name	Meaning
S, D, N_i	Node *ID*s, particularly, $S = source$, $D = Destination$
RREQ, *RREP* and *RERR*	The type identifier for a route request, a route reply and a route error, respectively
QNum and *RNum*	The randomly generated route request *ID* and the corresponding route reply *ID*, respectively. $RNum = QNum + 1$ for each route discovery
$HC_{i\to j}$	The hop-count from node N_i to N_j
QMAC, *RMAC* and *EMAC*	The k-MAC used in *RREQ*, *RREP* and *RERR*, respectively
$K_{i,j}$	The key shared between nodes N_i and N_j, thus $K_{i,j} = K_{j,i}$
$\{NodeList\}$	Records the intermediate nodes traversed by messages *RREQ*, *RREP* or *RERR*
$rT_{i\to j}$	The route from node N_i to node N_j

Because no node has been traversed by *RREQ* at the source S, $HC = 0$ and $\{NodeList\} = \{Null\}$:

$$QMAC_{s,d} = Hash(CORE, HC, \{NodeList\}, K_{s,d}) \quad (6)$$

is the k-MAC which will be further processed by intermediate nodes, and used by the destination D to verify the integrity of *RREQ* and the validity of the path recorded by $\{NodeList\}$. Parameter

$$CORE = Hash(RREQ, S, D, QNum, K_{s,d}) \quad (7)$$

serves as a credential of S to assure D that the *RREQ* is really originated from S and its immutable fields are integral during the propagation.

Route request forwarding: A *RREQ* received by an intermediate node N_i is processed and further broadcast only if it has never been seen (the *ID* of node S and the randomly generated *QNum* uniquely identify the current route discovery initialized by S). Because $\{NodeList\}$ records the nodes that have been traversed before the *RREQ* is received at N_i, N_i increases *HC* by one and appends the *ID* of the upstream node N_{i-1} into $\{NodeList\}$, and updates QMAC by

$$QMAC_{i,d} = Hash(QMAC_{i-1,d}, HC, \{NodeList\}, K_{i,d}). \quad (8)$$

A reverse forwarding entry is also established at N_i, which is used to relay the replied *RREP*s back to source S.

Check RREQ at destination D: Fig. 1 shows the verification procedure conducted at destination D. Basically, D repeats the computation executed by each intermediate node traversed by *RREQ*, which are recorded in field $\{NodeList\}$, *using the shared keys maintained by D itself*. Obviously, the number of hashing that D needs to perform equals *HC*, the number of nodes traversed by the *RREQ*. If such a verification is successful, D can be assured that the *RREQ* was really originated from S, each

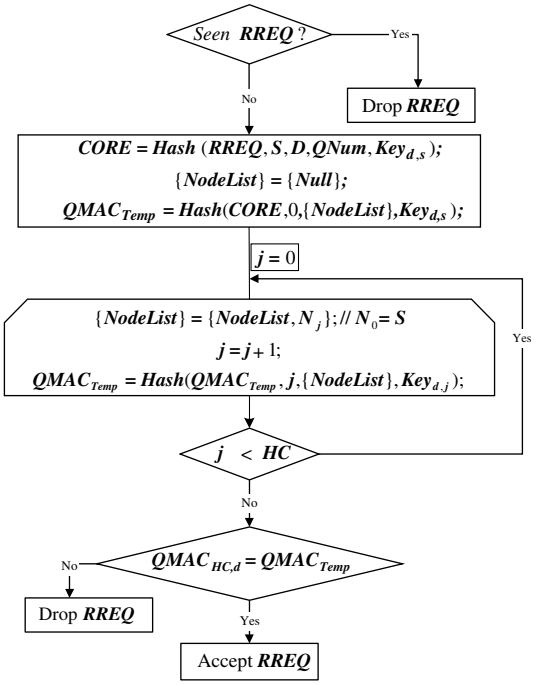

Fig. 1. Check *RREQ* at destination D.

node listed in $\{NodeList\}$ actually participated in the forwarding of *RREQ*, and the distance between S and D is equal to $HC_{s\to d}$.

The back-forwarding at intermediate nodes and the verification of *RREP* at the source S are basically symmetric to that of *RREQ*, and as such are omitted for brevity. Note that AOSR forwards traffic on a hop-by-hop basis, and each intermediate node relaying a *RREP* also establishes the forwarding entry for the requested destination D, which is used to route succeeding data packets.

Route maintenance: A route error message (*RERR*) is generated and unicast back to source S if an intermediate node N_i finds the downstream link of an active route is broken. Before accepting

a *RERR*, S must make sure that (a) the node generating the *RERR* belongs to the path for the destination; and (b) the node reporting link failure should actually be there when it was reporting the link failure. The process of sending back a *RERR* from node N_i is similar to that of originating a route reply from N_i to the source S. Therefore, here we only describe the main differences. A *RERR* has a format similar to that of a *RREP*, except the type identifier *RERR* and the initialization of *CORE*, which is calculated as follows:

$$CORE = Hash(RERR, N_i, S, D, RNum, K_{i,s}). \quad (9)$$

Each intermediate nodes in the reverse path to the source only processes and back-forwards a *RERR* received from its successor used for destination D, which ensures that no node rather than N_i can initialize a *RERR*, and node N_i is still in the path for D when reporting the link failure. When source S receives the *RERR*, it invokes a verification procedure similar to that of *RREP*. The only difference is the initial value of *CORE*, which is calculated by

$$CORE = Hash(RERR, N_i, S, D, RNum, K_{s,i}), \quad (10)$$

where rather than $K_{i,s}$ of node N_i, the pairwise key $K_{s,i}$ maintained at S is used.

5. Security analysis

The attacks to MANETs can be classified into *external attacks* and *internal attacks* based on the information acquired by the attackers. By having the access to keys owned by legal nodes, internal attacks are not as defensible as external attacks (the reader can refer to [6,7] for details of attack classification). Fig. 2 depicts the network topology used for our analysis. We only consider *RREQ* because the processing of *RREP* is symmetric.

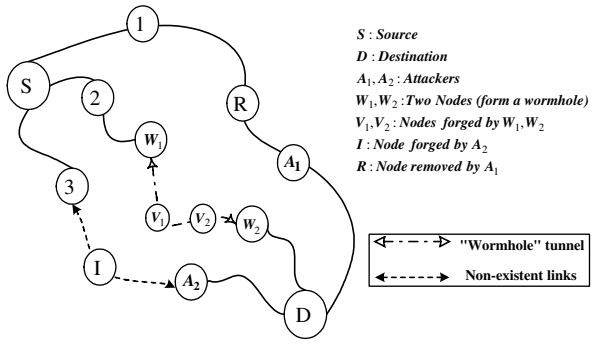

Fig. 2. Example network topology.

In AOSR, *RREQ* consists of immutable fields *RREQ*, *QNum*, S, D, and mutable fields *QMAC*, *HC* and {*NodeList*}. As to immutable parts, they are protected by the one-way hash value *CORE*, which has *RREQ*, S, D, *QNum* and $K_{s,d}$ as the input. No node can impersonate the initiator S to fabricate *RREQ* due to the lack of key $K_{s,d}$ known only to S and D. Any modification on such fields can be easily detected by destination D, because the *QMAC* carried in the *RREQ* cannot match what D recalculates based on {*NodeList*}.

Mutable fields *HC*, {*NodeList*} and *QMAC* are modified by intermediate nodes when the *RREQ* propagates to D. In AOSR, the authenticity of *HC*, {*NodeList*} and *QMAC* is guaranteed by integrating *HC* and {*NodeList*} into the computation of *QMAC*, in such a way that no node can be added into {*NodeList*} by the downstream node, unless it has actually forwarded a *RREQ*; and no node can be maliciously removed from {*NodeList*}, unless it is not used for routing traffic for D. For instance, let us assume that attacker A_1 attempts to remove node R from {*NodeList*} and decrease *HC* by one. When receiving the *RREQ*, D recomputes *QMAC* according to the nodes listed in {*NodeList*}. Because the hashing executed by R, i.e., $QMAC_{r,d}$, has been omitted, D cannot have a match with the received *QMAC*. The reason is that hashing operation is one-way only, and there is no way for A_1 to reverse the computation of $QMAC_{r,d}$. Another possible attack is for attacker A_2 to insert a non-existent node I into {*NodeList*} and increase *HC* by one. To achieve this, A_2 needs to perform one more hashing that requires $K_{i,d}$ as the input, which is impossible because $K_{i,d}$ is only known to I and D. For the same reason, A_2 cannot impersonate another node (Spoofing) and make itself appear on {*NodeList*}.

Wormhole is a special attack that is notoriously difficult to defend against. Wormhole usually consists of two nodes working collusively, picking up packets at one point of the network, tunneling them through an implicit channel, then releasing them at a faraway point. The goal is to mislead nodes near the releasing point to believe that the tunneled packets are transmitted by a nearby node. A demonstrative scenario of wormhole is shown in Fig. 3.

The chained k-MAC values computed by all intermediate nodes during the route discovery, together with the authenticated neighbor information, enable AOSR to detect wormhole and varied attacks derived from it. As an example, let us assume that nodes W_1 and W_2 in Fig. 2 are two

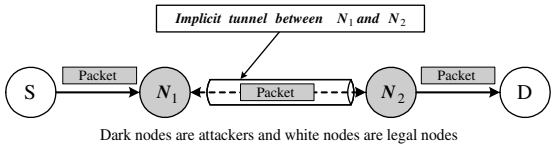

Fig. 3. Illustration of wormhole attack.

adversaries who have formed a tunnel $Tul_{w_1 \leftrightarrow w_2}$. First, they can refuse to forward *RREQ*, but this is not attractive because this actually excludes them from the route discovery. Second, they can attempt to modify *HC* or {*NodeList*}, but this can be detected when destination *D* checks the *QMAC* carried by *RREQ*. They can also insert some non-existent nodes, like V_1, V_2, into {*NodeList*}, but this cannot succeed due to the lack of shared keys $K_{v1,d}$ and $K_{v2,d}$.

Packets tunneled by external attackers can be detected because the MAC address of the outsider cannot match any *ID* maintained by the neighbor list at the receiving node near the releasing point (or does not exist at all). This can be done because a node's MAC address cannot be changed, any binding of a MAC address and an *ID* on the neighbor list has been authenticated, and the MAC address of a packet is always in clear text. For example, assume again that the nodes W_1 and W_2 in Fig. 2 are two external attackers and form a tunnel $Tul_{w_1 \leftrightarrow w_2}$, and w_1 or w_2 is tunneling a packet from node 2 to node *D*. This packet cannot be accepted because the MAC address shown in the packet (the MAC address of W_2) does not match the MAC address of node 2 maintained by node *D* (or, there is no neighbor entry maintained for node 2 at all).

Though there are other approaches to defending against wormhole attacks [8], time synchronization must be made available to each node for the proposed *packet leashes* to work. On the other hand, binding unalterable MAC address with nodal identifier is simple to implement and provides almost the same defensive results as packet leashes.

6. Simulation

We implement AOSR in NS2 [9], which acts as the centralized authority at the network formation, and provide time synchronization in the course of simulation. Simulation parameters are summarized in Table 2, and used throughout the following unless specified otherwise.

We present five metrics: (a) *packet delivery ratio (PDR)* is the number of CBR packets received averaged over the number of CBR packets originated; (b) *end-to-end packet delay* is the average elapsed time for a CBR packet to traverse between source and destination; (c) *route discovery delay* is the average time it takes for the source node to find a route for the requested destination; (d) *normalized routing overhead* is the total routing messages originated and forwarded over the total number of CBR packets received; and (e) *average route length* is the average length (hops) of the paths discovered.

Figs. 4 and 5 demonstrate the performance comparison between AOSR using S-NIKAP and the ad hoc on-demand distance vector routing protocol AODV [10]. When there is no attack occurring in the network, the normalized routing overhead of AOSR, as shown in Fig. 4(b), is almost the same as that of AODV. The reason is intuitive, establishing shared keys using NIKAP does not need the negotiation between nodes, or between the nodes and an on-line CA. In our simulation, the key progression interval is set to 5 s, and in practice, this is adjustable according to the processing power of mobile nodes, or the given security policy. Because shared keys between nodes need to be updated at a fixed rate, we expect that the time it takes for AOSR to discover routes should be longer than that of AODV. Fortunately, as shown in Fig. 5(a), the average routing delay caused by key progression,

Table 2
Simulation parameters

Parameter	Value
Topology	30 nodes, 1000 m × 250 m field and two-ray propagation model
MAC protocol	802.11 DCF, radio range = 250 m and bandwidth = 2×10^6 bits/s
Traffic pattern	15 constant bit rate (CBR) flows with randomly chosen source and destination, two packets per second, and with a payload size of 512 bytes. Each flow starts randomly within 50 s after the simulation is launched, and the lasting time varies between 100 and 200 s
Mobility model	Random way-point model with $V_{\min} = 0$ and $V_{\max} = 15$ m/s
Simulation time	300 s, 5 trials with different random seeds

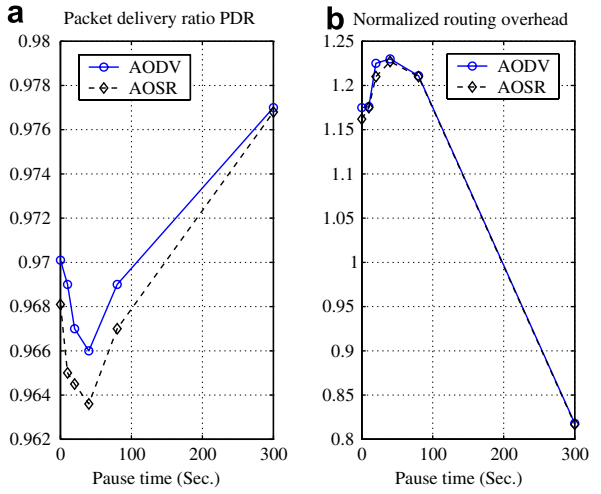

Fig. 4. PDR comparison w/o attacks.

measured over all nodes, is only 2–5 ms more than that of AODV, which is an acceptable increase of 5–12%. This indicates that NIKAP efficiently supports the security mechanisms used by the route-discovery process of AOSR without incurring significant routing delay. The average route length of AOSR is a little shorter than that of AODV, as shown in Fig. 5(c). The reason is that AOSR requires all route requests to reach the destination, while AODV allows intermediate nodes to reply to a RREQ if they cache an active route, which may not be the shortest at that moment. This also explains why the packet delivery delay of AOSR is shorter than that of AODV, as shown in Fig. 5(b).

Figs. 6 and 7 present the simulation results when 30% and 60% of the nodes in the network are compromised, and fabricate fake route replies to route requests by claiming that they are *zero* hop away from the specified destination node, in hope that the querying source node is willing to send its succeeding data packets to them. After that, a

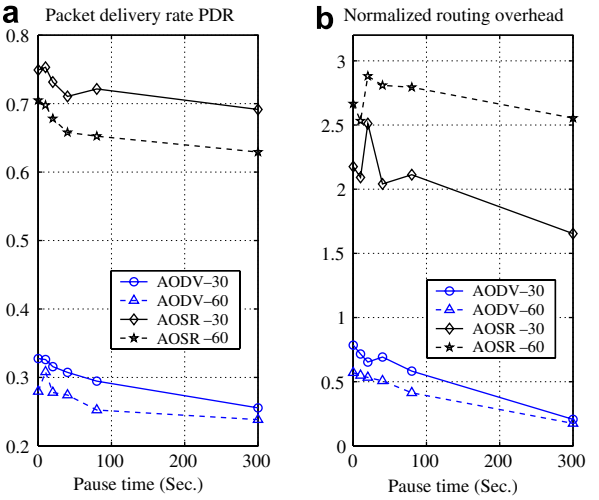

Fig. 6. PDR comparison with attacks.

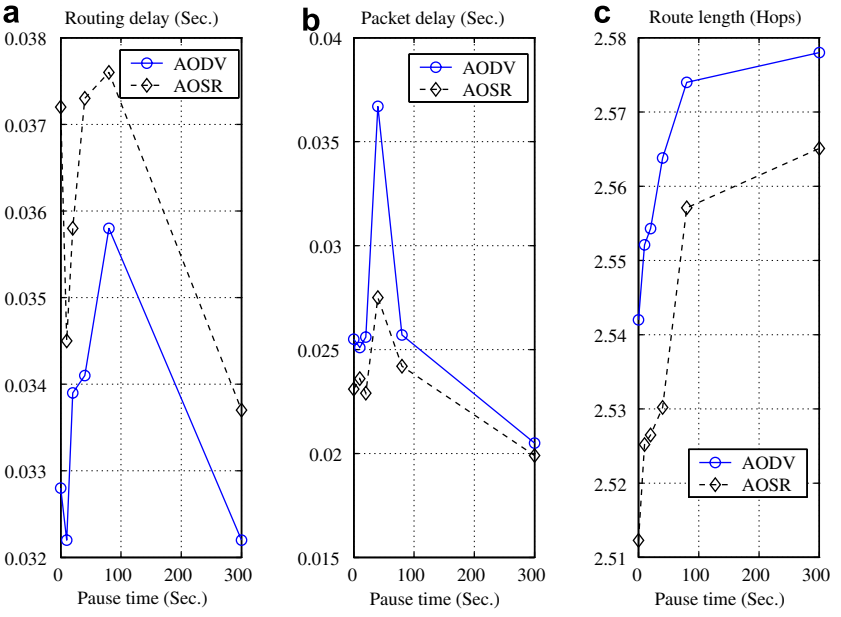

Fig. 5. Delay comparison w/o attacks.

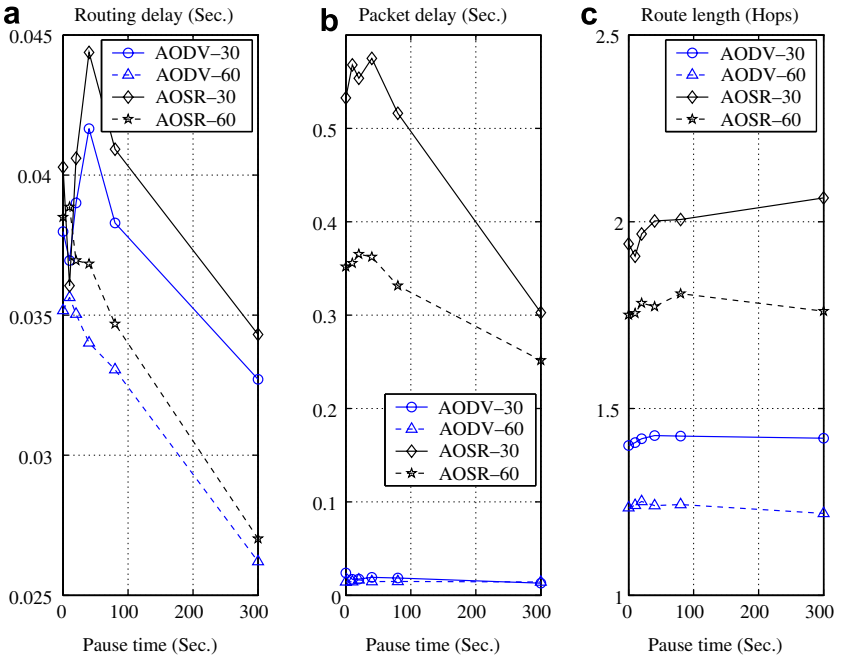

Fig. 7. Delay comparison with attacks.

compromised node simply drops all the data packets received (black-hole attack).

The packet delivery ratio of AODV decreases drastically, as shown in Fig. 6(a), given that most of the packets are sent to the compromised nodes, which discard them silently. The average route length of AODV is much shorter than when there is no malicious node in the network, as shown in Fig. 7(c). The reason is that a compromised node is likely to receive and reply to the route requests for the specified destination earlier than the destination itself, or other nodes having an active route. This also indicates that most of the successful packets are delivered within one or two hops away from the source.

As shown in Fig. 6(a), AOSR is still able to sustain over 62% packet delivery ratios for all pause time configurations, even when 60% of the nodes are compromised. This is achieved at the cost of more time to find a route, longer end-to-end packet delay and higher overhead, as shown in Figs. 7(a), (b), and 6(b), respectively. Lastly, nodes running AOSR cannot be misled by compromised nodes declaring better reachability for the requested destination (AOSR detects the misbehavior of malicious nodes when the verification of *RREQ* or *RREP* fails), and as such are able to find a route to the destination if there is one. Consequently, the average length of routes discovered by AOSR is longer than that of AODV, as shown in Fig. 7(c).

7. Conclusion

We proposed S-NIKAP and A-NIKAP, two key agreement protocols that achieves non-interactive key establishment and if needed, the succeeding key progression (rekeying process). NIKAP needs the aid of a centralized authority only at the initial network formation, which is better than other approaches relying on on-line CA services.

References

[1] J. Kohl, B. Neuman, The Kerberos Network Authentication Service (V5), RFC 1510, September, 1993.
[2] L. Zhou, Z.J. Haas, Securing ad hoc networks, IEEE Network 13 (6) (1999) 24–30, Special Issue on Network Security.
[3] W. Diffie, E. Hellman, New directions in cryptography, IEEE Trans. Inform. Theory 22 (1976) 644–654.
[4] H. Petersen, P. Horster, Self-certified keys – concepts and applications, in: Third Conference of Communications and Multimedia Security, Athens, September 1997.
[5] Z. Li, J.J. Garcia-Luna-Aceves, Enhancing the security of on-demand routing in ad hoc networks, in: AdhocNow'05, Cancun, Mexico, October 2005.
[6] K. Sanzgiri, B. Dahill, B.N. Levine, E. Royer, C. Shields, A secure routing protocol for ad hoc networks, in: Tenth Conference on Network Protocols (ICNP), 2002.

[7] Y. Hu, A. Perrig, D. Johnson, Ariadne: a secure on-demand routing protocol for ad hoc networks, MobiCom'02, September 2002.
[8] Y. Hu, A. Perrig, D. Johnson, Packet leashes: a defense against wormhole attacks in wireless networks, in: IEEE INFOCOM, San Francisco, US, March 2003.
[9] NS2, the Network Simulator. Available from: <http://www.isi.edu/nsnam/ns/>.
[10] C. Perkins, E. Royer, S. Das, Ad hoc on demand distance vector (AODV) routing, RFC 3561 (Experimental), July 2003.
[11] C. Castelluccia, N. Saxena, J.H. Yi, Self-configurable key pre-distribution in mobile ad-hoc networks, in: IFIP Networking Conference, Waterloo, Canada, May 2005.
[12] E. Pagani, Providing reliable and fault tolerant broadcast delivery in mobile ad-hoc networks, Mob. Network Appl. 4 (3) (1999) 175–192.

Zhenjiang Li received the B.S. and M.S. degrees in electronic engineering from University of Science and Technology of China (USTC), Hefei, China, in 1998 and 2001, respectively. Since 2001, he has been a Ph.D. student of the computer communication research group (CCRG) in department of computer engineering, University of California, Santa Cruz, US. His research interests include multi-constrained path computation for QoS routing in computer networks, key distribution and management in wireless networks, secure ad hoc routing and formal verification techniques of secure routing protocols. He is a student member of the IEEE and ACM.

J.J. Garcia-Luna-Aceves received the B.S. degree in electrical engineering from the Universidad Iberoamericana in Mexico City, Mexico in 1977, and the M.S. and Ph.D. degrees in electrical engineering from the University of Hawaii, Honolulu, HI, in 1980 and 1983, respectively. He holds the Jack Baskin Chair of Computer Engineering at the University of California, Santa Cruz (UCSC), and is a Principal Scientist at the Palo Alto Research Center (PARC). Prior to joining UCSC in 1993, he was a Center Director at SRI International (SRI) in Menlo Park, California. He has been a Visiting Professor at Sun Laboratories and a Principal of Protocol Design at Nokia.

He has published a book, more than 300 papers, and 17 US patents. He has directed 25 Ph.D. theses and 20 M.S. theses since he joined UCSC in 1993. He has been the General Chair of the IEEE SECON 2005 Conference; Program Co-Chair of ACM MobiHoc 2002 and ACM Mobicom 2000; Chair of the ACM SIG Multimedia; General Chair of ACM Multimedia'93 and ACM SIGCOMM'88; and Program Chair of IEEE MULTIMEDIA'92, ACM SIGCOMM'87, and ACM SIGCOMM'86. He has served in the IEEE Internet Technology Award Committee, the IEEE Richard W. Hamming Medal Committee, and the National Research Council Panel on Digitization and Communications Science of the Army Research Laboratory Technical Assessment Board. He has been on the editorial boards of the IEEE/ACM Transactions on Networking, the Multimedia Systems Journal, and the Journal of High Speed Networks. He received the SRI International Exceptional-Achievement Award in 1985 and 1989, and is a Fellow of the IEEE.