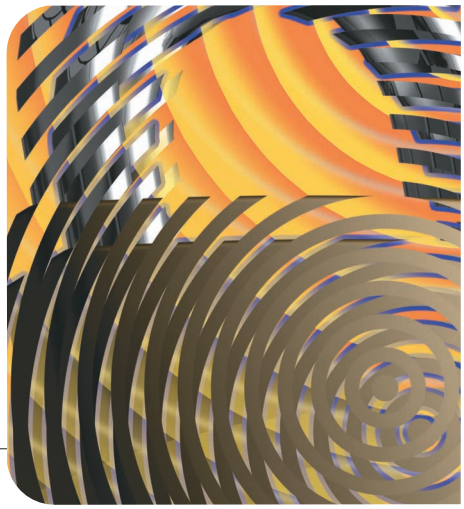
A Survey of Secure Wireless Ad Hoc Routing

Ad hoc networks use mobile nodes to enable communication outside wireless transmission range. Attacks on ad hoc network routing protocols disrupt network performance and reliability. The authors survey the state of research and its challenges in this field.



Yih-Chun Hu
University of California, Berkeley

Adrian Perrig
Carnegie Mellon University

In a multihop wireless ad hoc network, mobile nodes cooperate to form a network without using any infrastructure such as access points or base stations. Instead, the mobile nodes forward packets for each other, allowing communication among nodes outside wireless transmission range. The nodes' mobility and the fundamentally limited capacity of the wireless medium, together with wireless transmission effects such as attenuation, multipath propagation, and interference, combine to create significant challenges for routing protocols operating in an ad hoc network.

Examples of applications for ad hoc networks range from military operations and emergency disaster relief to community networking and interaction among meeting attendees or students during a lecture. In these and other ad hoc networking applications, security in the routing protocol is necessary to guard against attacks such as malicious routing misdirection.

This article reviews attacks on ad hoc networks and discusses current approaches for establishing cryptographic keys in ad hoc networks. We describe the state of research in secure ad hoc routing protocols and its research challenges.

Attacks on ad hoc networks

Attacks on ad hoc network routing protocols generally fall into one of two categories:

- *Routing-disruption attacks.* The attacker attempts to cause legitimate data packets to be routed in dysfunctional ways.
- *Resource-consumption attacks*. The attacker injects packets into the network in an attempt to consume valuable network resources such as bandwidth or to consume node resources such as memory (storage) or computation power.

From an application-layer perspective, both attacks are instances of a denial-of-service (DoS) attack.

An example of a routing-disruption attack is for an attacker to send forged routing packets to create a routing loop, causing packets to traverse nodes in a cycle without reaching their destinations, thus consuming energy and available bandwidth. An attacker might similarly create a routing *blackhole*, which attracts and drops data packets. An attacker creates a blackhole by distributing forged routing information (that is, claiming falsified short distance information); the attacker attracts traffic and can then discard it. In a special case of a black hole, an attacker could create a *gray hole*, in which it selectively drops some packets but not others, for example, by forwarding routing packets but not data packets. An attacker also might attempt to cause a node to use a route detour (suboptimal routes), or partition the network by injecting forged routing information to prevent one set of nodes from reaching another. An attacker might attempt to make a route through itself appear longer by adding virtual nodes to the route; we call this attack a *gratuitous detour* because a shorter route exists and would otherwise have been used. In ad hoc network routing protocols that attempt to keep track of perceived malicious nodes in a blacklist at each node, such as is done in the watchdog and pathrater protocol,[1] an attacker might malign a good node, causing other good nodes to add that node to their blacklists and thus avoid that node in future routes.

A more subtle type of routing-disruption attack is cre-

ating a *wormhole* in the network, using a pair of attacker nodes *A* and *B* linked via a private network connection. We give an example of this attack and a countermeasure in the next section.

A *rushing* attack is a malicious attack that is targeted against on-demand routing protocols that use duplicate suppression at each node.[2] An attacker disseminates `ROUTE REQUESTS` quickly throughout the network, suppressing any later legitimate `ROUTE REQUESTS` when nodes drop them due to the duplicate suppression.

Packet leashes

The wormhole attack[3] is a severe attack against ad hoc routing protocols that is particularly challenging to defend against; it can potentially cripple a range of ad hoc network routing protocols. In the wormhole attack, an attacker records packets (or bits) at one location in the network, tunnels them to another location, and retransmits them from there into the network. Most existing ad hoc network routing protocols that lack a mechanism to defend them against the wormhole attack would be unable to find routes longer than one or two hops, which severely disrupts communication.

If a wormhole attacker tunnels all packets through the wormhole honestly and reliably, no harm is done; the attacker actually provides a useful service in connecting the network more efficiently. However, when an attacker forwards only routing control messages, this attack might severely disrupt routing. For example, when used against an on-demand routing protocol such as DSR[4] or AODV,[5] a powerful application of the wormhole attack can be mounted by tunneling each `ROUTE REQUEST` packet directly to the target node of the `REQUEST`. This attack prevents any node from discovering routes more than two hops long.

Periodic protocols are also vulnerable to this kind of attack. For example, OLSR[6] and TBRPF[7] use `HELLO` packets for neighbor detection, so if an attacker tunnels to *B* all `HELLO` packets transmitted by *A* and tunnels to *A* all `HELLO` packets transmitted by *B*, then *A* and *B* will believe that they are neighbors, which would cause the routing protocol to fail to find routes when they aren't actually neighbors.

The wormhole attack is also dangerous in other wireless applications. One example is any wireless access control system that is proximity based, such as wireless car keys or proximity- and token-based access control systems for PCs.[8,9] In such systems, an attacker could relay authentication exchanges to gain unauthorized access.

Our solution to the wormhole attack is *packet leashes*. We consider specifically two types of packet leashes: *geographical* and *temporal*. The main idea is that by authenticating either an extremely precise timestamp or location information combined with a loose timestamp, a receiver can determine if the packet has traversed an unrealistic distance for the specific network technology used. Temporal leashes rely on extremely precise time synchronization and timestamps in each packet. We can approximate a packet's travel time as the difference between the receive time and the timestamp. To be more conservative, however, a node can choose to add the maximum time synchronization error, assuming that the sender's clock might be faster than the receiver's. Conversely, to allow all direct communication between legitimate nodes, a node can subtract the maximum time synchronization error, assuming that the sender's clock might be slower than the receiver's.

Given the precise time synchronization required by temporal leashes, we constructed some efficient broadcast authenticators based entirely on symmetric primitives. In particular, we extend the Timed Efficient Stream Loss-Tolerant Authentication (Tesla) broadcast authentication protocol[10] to allow the disclosure of the authentication key within the authenticated packet. We use a Merkle tree[11] to authenticate these keys. Our research shows that with this authentication mechanism, currently available devices easily can support line-speed authentication of temporal leashes.

Another method of constructing a leash is to use location information and loosely synchronized clocks. We call such leashes *geographical leashes*. If the sender and receiver clocks are synchronized to within $\pm\Delta$, and v is an upper bound on any node's velocity, then the receiver can compute an upper bound on the distance between the sender and itself d_{sr}. Specifically, based on the timestamp t_s in the packet, the local receive time t_r, the maximum relative error in location information δ, and the locations of the receiver p_r and the sender p_s, d_{sr} can be bounded by $d_{sr} \leq \|p_s - p_f\| + 2v \cdot (t_r - t_s + \Delta) + \delta$.

In certain circumstances, bounding the distance between the sender and receiver d_{sr} can't prevent wormhole attacks; for example, when obstacles prevent communication between two nodes that would otherwise be in transmission range, a distance-based scheme would still allow wormholes between the sender and receiver. A network that uses location information as a leash can control even these kinds of wormholes, though. To accomplish this, each node has a radio propagation model. A receiver verifies that every possible sender location (a δ +

Given the precise time synchronization, we constructed some efficient broadcast authenticators based entirely on symmetric primitives.

$v(t_r - t_s + 2\Delta)$ radius around p_s) can reach every possible receiver location (a $(\delta + v(t_r - t_s + 2\Delta)$ radius around p_r).

Private-key distribution is more challenging than public-key distribution because protocols for key distribution must ensure the secrecy of such keys.

Attacker model

In previous work,[12] we defined an attacker model consisting of two main attacker classes, passive and active. A *passive attacker* doesn't send messages; it just eavesdrops on the network. Passive attackers are mainly threats against communication privacy or anonymity, rather than against the network's function or routing protocol, and thus we don't discuss them further here.

An *active attacker* injects packets into the network (but it eavesdrops as well). We characterize this kind of attacker based on the number of nodes it owns in the network and the number of good nodes it has compromised. We assume that the attacker owns all the cryptographic key information of compromised nodes and distributes it among all its nodes. We denote such an attacker Active-*n*-*m*, where *n* is the number of nodes it has compromised and *m* is the number of nodes it owns. We propose the following attacker hierarchy (with increasing strength) to measure routing protocol security: Active-0-1 (the attacker owns one node), Active-0-*x* (the attacker owns *x* nodes), Active-1-*x* (the attacker owns one compromised node and distributes the cryptographic keys to its $x - 1$ other nodes), and Active-*y*-*x*. In addition, we call an attacker that has compromised nodes an ActiveVC attacker if it owns all nodes on a vertex cut through the network that partitions the good nodes into multiple sets, forcing good nodes in different partitions to communicate only through an attacker node. This attack is particularly powerful because it controls all traffic between nodes of the disjoint partitions.

Key setup in ad hoc networks

In many ad hoc networks, the compromise of a single network node and the capture of its cryptographic keys is a viable threat. Intuitively, a single compromised node is less powerful than numerous compromised nodes; in future sections, we discuss several protocols that can limit a single compromised node's effectiveness.

To achieve this higher level of security against single compromised nodes, however, it must be possible for the routing protocol to distinguish among the several legitimate nodes. Such nodes can be distinguished through the use of authentication, but a single shared key can't provide authentication; instead, each legitimate node must possess one or more keys unique to that node. In addition, each node must have a way to authenticate a legitimate node. How to disseminate authentic key information is the key-setup problem, and researchers have proposed a number of solutions to it.

Establishing private keys

Some ad hoc network routing protocols require shared private keys between all pairs of nodes in the network. Private-key distribution is substantially more challenging than public-key distribution because protocols for key distribution must ensure the secrecy of such keys.

The obvious way to distribute private keys is to share them with each pair of nodes before deployment, when we know all nodes are behaving correctly. This approach is more difficult when incremental deployment of network nodes is desirable. Frank Stajano and Ross Anderson[13] propose a scheme for establishing trust and keys between two nodes in an ad hoc network; in their resurrecting duckling model, two nodes are touched together to bind a slave node to a master node. Once a key is exchanged through this physical link, that key can be used to encrypt and authenticate further information, such as a list of shared keys. Dirk Balfanz and his colleagues[14] generalize their approach, assuming a side channel in which it's possible to detect multiple transmitters.

If public keys have already been established, we can establish private keys by using a key-exchange protocol.[15] In future sections, we discuss approaches specific to distributing public keys—that is, keys that are distributed to all nodes. We can use such keys to verify authenticated messages but not to generate forged authenticators.

Avoiding the problem

One approach for solving the key-management problem is to assume that each node carries a list of legitimate public keys. This approach is by far the most straightforward; however, it assumes that all nodes trust a common set of authorities and that each node can download a list of legitimate nodes before deployment.

Another issue with this approach concerns incremental deployment. If network nodes are not deployed nearly simultaneously, then one node might be deployed before a future node can provide its keys to the authority. In this case, the authority would need to generate keys for future nodes. When a new node wants to join the network, it receives both the list of legitimate nodes as well as its own private key. In this case, the channel over which it receives the private key must be secure against eavesdropping. Ordinarily, the channel over which it receives the list of legitimate nodes would need to be secure only against ac-

tive attacks; such security could be provided by using a side channel to check the equality of a cryptographic hash function computed over the list.

SUCV addresses

Gabriel Montenegro and Claude Castelluccia propose an approach in which each node chooses an address based on its public key.[16] In this approach, called statistically unique cryptographically verifiable (SUCV) addresses, each node generates a public- and private-key pair, and then chooses its address based on a cryptographic hash function of the public key. The authors proposed two schemes: one in which a node's entire IPv6 address was the hash function's output and another in which the least significant 64 bits were the hash function's output. Each node generates its own public- and private-key pair. As a result, that node can find another key that hashes to the same 64-bit address in 2^{32} time, but any other node must on average do 2^{63} work to find such a collision.

Unfortunately, this approach doesn't entirely solve the key-setup problem. In particular, in a network without SUCV, the problem is to have a list of legitimate (node, public-key) pairings; in a network using SUCV, the problem is to have a list of legitimate nodes.

Certificates from a certificate authority

Another approach is to define one or more certificate authorities (CAs). Each node in the network has a certificate that includes its node address, its public key, and a signature from the CA. If each node includes its certificate each time it signs a message, the recipient can first verify the certificate, and then use the public key in the certificate to check the signature. Optimizations might let the protocol include the certificate less frequently.

CAs can be online or offline. The difference between these types is whether they are reachable through the network. Because an online CA is reachable through the network, it can participate in the certificate verification protocol. However, even with an online CA, the certificate verification protocol might not be straightforward because there is a circular dependency between security and routing: to run the security protocol, routes must be established to the CA, but to establish these routes, a routing protocol first must function. In Ariadne, we broke this circular dependency[12] by using the CA, which is already trusted, to establish these routes.

An online CA is often vulnerable to compromise. Lidong Zhou and Zygmunt J. Haas propose using threshold cryptography to distribute the CA functionality to several nodes.[17] One simple way of achieving this is to only trust keys that have certifications from several CAs. This technique allows the compromise of one or two nodes participating in the certifications without letting an attacker certify an arbitrary number of new nodes. We can use threshold cryptography to provide this same semantic, which has the advantage of providing a smaller certificate. Seung Yi and Robin Kravets provide a more complete discussion of protocols for using CAs in ad hoc networks.[18]

Transitive trust and PGP trust graphs

Researchers have proposed an alternative trust model for ad hoc networks without an online CA. In this model, each node signs certificates for other nodes.[19] A node can search the network for a chain of certificates leading from the node initiating the query and ending at the node trying to authenticate a message. Generally, such schemes require transitive trust—that is, if *A* trusts *B*, and *B* trusts *C*, then such schemes require that *A* trust *C*.

This requirement can be slightly relaxed by requiring additional node-disjoint paths and limiting the certificate chain's length. If multiple node-disjoint paths exist between the signer and verifier, then an attacker would have had to compromise one node on each path to insert a bogus key. When the certificate's length is limited, the impact of a compromised node can be limited. For example, a network can limit certificate chains to one intermediate signer. In this case, if no neighbor of a node *A* is compromised, then an attacker can't generate a bogus certificate that *A* would accept.

Public-key revocation

Ideally, when a node is compromised, some authority can revoke the certificate for its public key. Any such system would need to guard against an attacker that attempts to revoke the keys of legitimate nodes.

Setting up these revocations is a problem even more difficult than the key-setup problem. One way such certificates can be revoked is to use an online CA to sign negative certificates. However, a node would generally be unwilling to distribute its own negative certificate. As a result, an alternative distribution mechanism is necessary. We proposed an approach where blacklist or other revocation information can be flooded through the network when it is discovered.[2] With this approach, some flood limiting is necessary, or an attacker can get its certificate revoked and repeatedly flood the network with this revocation information. As a result, we specified that only new revocation information be flooded.

SEAD in mobile wireless ad hoc networks

Our Secure Efficient Ad hoc Distance vector routing protocol (SEAD)[20] is robust against multiple uncoordinated attackers creating incorrect routing state in any other node, in spite of active attackers or compromised nodes in the network. We based SEAD's design in part on the Destination-Sequenced Distance-Vector ad hoc network routing protocol (DSDV).[21] To support use of SEAD with nodes of limited CPU processing capability,

and to guard against DoS attacks in which an attacker attempts to cause other nodes to consume excess network bandwidth or processing time, we use efficient one-way hash functions and don't use asymmetric cryptographic operations in the protocol.

Distance-vector routing

In distance-vector routing, each router maintains a routing table listing all possible destinations within the network. Each entry in a node's routing table contains the address (identity) of some destination, the node's shortest known distance (called the *metric*, usually in number of hops) to the destination, and the address of the node's neighbor router that is the first hop on the shortest route to the destination. The distance to the destination is a metric in that table entry. Each router forwarding a packet uses its own routing table to determine the next hop toward the destination.

To maintain the routing tables, each node periodically broadcasts a routing update containing the information from its own routing table. Each node updates its own table using the updates it hears so that its route for each destination uses as a next hop the neighbor that advertised the smallest metric in its update for that destination. The node sets the metric in its table entry for that destination to one (hop) more than the metric in that neighbor's update.

The primary improvement for ad hoc networks made in DSDV over standard distance vector routing is the addition of a sequence number in each routing table entry. Using this sequence number prevents routing loops caused by updates being applied out of order. This problem can be common over multihop wireless transmission because the routing information can spread along many different paths through the network.

Hash chains

A one-way hash chain is built on a one-way hash function. Like a normal hash function, a one-way hash function H maps an input of any length to a fixed-length bit string. Thus, H: $\{0,1\}^{\star} \rightarrow \{0,1\}^{\rho}$, where ρ is the length in bits of the hash function's output. The function H should be simple to compute yet must be computationally infeasible in general to invert.

To create a one-way hash chain, a node chooses a random $x \in \{0,1\}^{\rho}$ and computes the list of values h_0, h_1, h_2, h_3, ..., h_n,where $h_0 = x$, and $h_i = H(h_{i-1})$ for $0 < i \leq n$, for some n. The node at initialization generates the elements of its hash chain using this recurrence, in order of increasing subscript i; over time, it uses certain elements of the chain to secure its routing updates. In using these values, the node progresses in order of decreasing subscript i within the generated chain.

Given an existing authenticated element of a one-way hash chain, we can verify elements later in the sequence of use within the chain (further on, in order of decreasing subscript). For example, given an authenticated h_i value, a node can authenticate h_{i-3} by computing $H(H(H(h_{i-3})))$ and verifying that the resulting value equals h_i. To use one-way hash chains for authentication, we assume some mechanism for a node to distribute an authentic element such as h_n from its generated hash chain.

Authenticating routing updates

Each node in SEAD uses a specific single next element from its hash chain in each routing update that it sends about itself (metric 0). Based on this initial element, the one-way hash chain conceptually provides authentication for the metric's lower bound in other routing updates for this destination; the authentication provides only a lower bound on the metric—that is, an attacker can increase the metric or claim the same metric, but can't decrease the metric.

We assume that the network operator can place an upper bound on the ad hoc network's diameter, and we use $m - 1$ to denote this bound. The method SEAD uses to authenticate an entry in a routing update uses the sequence number in that entry to determine a contiguous group of m elements from that destination node's hash chain, one element of which must be used to authenticate that routing update. The particular element from this group of elements that must be used to authenticate the entry is determined by the metric value being sent in that entry. Specifically, if a node's hash chain is the sequence of values h_0, h_1, h_2, h_3, ..., h_n and n is divisible by m, then for a sequence number i in some routing update entry, let

$$k = \frac{n}{m} - i \ .$$

An element from the group of elements h_{km}, h_{km-1}, …, h_{km+m-1} from this hash chain is used to authenticate the entry; if the metric value for this entry is j, $0 \leq j < m$, then the value h_{km-j} is used here to authenticate the routing update entry for that sequence number. Nodes receiving any routing update easily can authenticate each entry in the update, given any earlier authentic hash element from the same hash chain.

A secure on-demand routing protocol for ad hoc networks

Ariadne[12] is a secure on-demand routing protocol that withstands node compromise and relies only on highly efficient symmetric cryptography. Ariadne can authenticate routing messages using one of three schemes:

- shared secrets between each pair of nodes,
- shared secrets between communicating nodes combined with broadcast authentication, or
- digital signatures.

In this article, we primarily discuss using Ariadne with Tesla,[10] an efficient broadcast authentication

scheme that requires loose time synchronization. Using pair-wise shared keys avoids the need for time synchronization but at the cost of higher key-setup overhead. Ariadne's basic idea is based on DSR:[4] Ariadne discovers routes on-demand (as they are needed) through route discovery and uses them to source route data packets to their destinations. Each forwarding node helps by performing route maintenance to discover problems with each selected route.

Basic Ariadne route discovery

We present the Ariadne protocol's design in two stages: we first present a mechanism that lets the target verify the authenticity of the `ROUTE REQUEST` and then present an efficient per-hop hashing technique to verify that no node is missing from the node list in the `REQUEST`. In the following discussion, we assume that the initiator S performs a route discovery for target D and that they share the secret keys K_{SD} and K_{DS}, respectively, for message authentication in each direction.

Target authenticates route requests. To convince the target of the legitimacy of each field in a `ROUTE REQUEST`, the initiator simply includes a message authentication code (MAC) computed with key K_{SD} over unique data—for example, a timestamp. The target can easily verify the route request's authenticity and freshness using the shared key K_{SD}.

In a route discovery, the initiator wants to authenticate each individual node in the node list of the `ROUTE REPLY`. A secondary requirement is that the target can authenticate each node in the node list of the `ROUTE REQUEST` so that it will return a `ROUTE REPLY` only along paths that contain legitimate nodes. Each hop authenticates the new information in the `REQUEST` using its current Tesla key. The target buffers the `REPLY` until intermediate nodes can release the corresponding Tesla keys. The Tesla security condition is verified at the target, and the target includes a MAC in the `REPLY` to certify that the security condition was met.

Per-hop hashing. Authenticating data in routing messages isn't sufficient because an attacker could remove a node from the node list in a `REQUEST`. We use one-way hash functions to verify that no hop was omitted, an approach we call per-hop hashing. To change or remove a previous hop, an attacker must either hear a `REQUEST` without that node listed or must be able to invert the one-way hash function. For efficiency, we can include the authenticator in the hash value passed in the `REQUEST`. Figure 1 shows an example of Ariadne route discovery.

Basic Ariadne route maintenance

Route maintenance in Ariadne is based on DSR. A node forwarding a packet to the next hop along the source route returns a `ROUTE ERROR` to the packet's original sender if it is unable to deliver the packet to the next hop after a limited number of retransmission attempts. This section discusses mechanisms for securing `ROUTE ERRORS` but doesn't consider the case of attackers not sending `ERRORS`.

To prevent unauthorized nodes from sending `ERRORS`, we require that the sender authenticate an `ERROR`. Each node on the return path to the source forwards the `ERROR`. If the authentication is delayed—for example, when Tesla is used—each node that will be able to authenticate the `ERROR` buffers it until it can be authenticated.

S:	$h_0 = \mathrm{MAC}_{K_{SD}}(\texttt{REQUEST}, S, D, id, ti)$
$S \rightarrow *$:	$\texttt{REQUEST}, S, D, id, ti, h_0, (), ()\rangle$
A:	$h_1 = H[A, h_0]$
	$M_A = \mathrm{MAC}_{K_{A_{ti}}}\texttt{REQUEST}, S, D, id, ti, h_1, (A), ())$
$A \rightarrow *$:	$\texttt{REQUEST}, S, D, id, ti, \mathbf{h_1}, (\mathbf{A}), \mathbf{M_A})$
B:	$h_2 = H[B, h_1]$
	$M_B = \mathrm{MAC}_{K_{B_{ti}}}(\texttt{REQUEST}, S, D, id, ti, h_2, (A, B), (M_A))$
$B \rightarrow *$:	$\texttt{REQUEST}, S, D, id, ti, \mathbf{h_2}, (A, \mathbf{B}), (M_A, \mathbf{M_B})\rangle$
C:	$h_3 = H[C, h_2]$
	$M_C = \mathrm{MAC}_{K_{C_{ti}}}(\texttt{REQUEST}, S, D, id, ti, h_3, (A, B, C), (M_A, M_B))$
$C \rightarrow *$:	$\texttt{REQUEST}, S, D, id, ti, \mathbf{h_3}, (A, B, \mathbf{C}), (M_A, M_B, \mathbf{M_C})$
D:	$M_D = \mathrm{MAC}_{K_{DS}}(\texttt{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C))$
$D \rightarrow C$:	$\texttt{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), \mathbf{M_D}, ()$
$C \rightarrow B$:	$\texttt{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (\mathbf{K_{C_{ti}}})$
$B \rightarrow A$:	$\texttt{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (K_{C_{ti}}, \mathbf{K_{B_{ti}}})$
$A \rightarrow S$:	$\texttt{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (K_{C_{ti}}, K_{B_{ti}}, \mathbf{K_{A_{ti}}})$

Figure 1. Route discovery example in Ariadne. The initiator node S is attempting to discover a route to the target node D. The bold font indicates changed message fields, relative to the previous message of that type.

Avoiding routing misbehavior

Ariadne, as we have described it so far, is vulnerable to an attacker that happens to be along the discovered route. In particular, we haven't presented a means of determining whether intermediate nodes are in fact forwarding packets that they have been requested to forward. To avoid the continued use of malicious routes, we choose routes based on their prior performance in packet delivery. Our scheme relies on feedback about which packets were successfully delivered. The feedback can be received either through an extra end-to-end network layer message or by exploiting properties of transport layers, such as TCP with Selective Acknowledgments.[22] This feedback approach is somewhat similar the one used in IPv6 for neighbor unreachability detection.[23]

A node with multiple routes to a single destination can assign a fraction of packets that it originates to be sent

$S \rightarrow *$: (ROUTE REQUEST, D, cert_S, N, $t)_{K_S^-}$
$A \rightarrow *$: ((ROUTE REQUEST, D, cert_S, N, $t)_{K_S^-})_{K_A^-}$, cert_A
$B \rightarrow *$: ((ROUTE REQUEST, D, cert_S, N, $t)_{K_S^-})_{K_B^-}$, cert_B
$C \rightarrow *$: ((ROUTE REQUEST, D, cert_S, N, $t)_{K_S^-})_{K_C^-}$, cert_C
$D \rightarrow C$: ((ROUTE REPLY, S, cert_D, N, $t)_{K_D^-}$
$C \rightarrow B$: ((ROUTE REPLY, S, cert_D, N, $t)_{K_D^-})_{K_C^-}$, cert_C
$B \rightarrow A$: ((ROUTE REPLY, S, cert_D, N, $t)_{K_D^-})_{K_B^-}$, cert_B
$A \rightarrow S$: ((ROUTE REPLY, S, cert_D, N, $t)_{K_D^-})_{K_A^-}$, cert_A

Figure 2. Route discovery in ARAN. In this figure, node S is discovering a route to node D. Each node rebroadcasts the first route request packet it receives from each route discovery. When the route request reaches the target, the destination returns a route reply to the node from which it heard that route request. Each node hearing a route reply forwards the reply to the node from which it heard the request.

along each route. When a substantially smaller fraction of packets sent along any particular route is successfully delivered, the node can begin sending a smaller fraction of its overall packets to that destination along that route.

Lightweight security for DSR

Panagiotis Papadimitratos and Zygmunt Haas[24] propose the Secure Routing Protocol, which we can use with DSR or the Interzone Routing Protocol in the Zone Routing Protocol (ZRP).[25] They designed SRP as an extension header that is attached to ROUTE REQUEST and ROUTE REPLY packets. SRP doesn't attempt to secure ROUTE ERROR packets but instead delegates the route-maintenance function to the Secure Route Maintenance portion of the Secure Message Transmission protocol. SRP uses a sequence number in the REQUEST to ensure freshness, but this sequence number can only be checked at the target. SRP requires a security association only between communicating nodes and uses this security association just to authenticate ROUTE REQUESTS and ROUTE REPLYS through the use of message authentication codes. At the target, SRP can detect modification of the ROUTE REQUEST, and at the source, SRP can detect modification of the ROUTE REPLY.

However, SRP doesn't attempt to prevent unauthorized modification of fields that are ordinarily modified in the course of forwarding these packets. For example, a node can freely remove or corrupt the node list of a ROUTE REQUEST packet that it forwards.

Because SRP requires a security association only between communicating nodes, it uses extremely lightweight mechanisms to prevent other attacks. For example, to limit flooding, nodes record the rate at which each neighbor forwards ROUTE REQUEST packets and gives priority to REQUEST packets sent through neighbors that less frequently forward REQUEST packets. Such mechanisms can secure a protocol when few attackers are present; however, such techniques provide secondary attacks, such as sending forged ROUTE REQUEST packets to reduce the effectiveness of a node's authentic ROUTE REQUESTS. In addition, such techniques exacerbate the problem of greedy nodes. For example, a node that doesn't forward ROUTE REQUEST packets ordinarily achieves better performance because it is generally less congested, and it doesn't need to use its battery power to forward packets originated by other nodes. In SRP, a greedy node retains these advantages and, in addition, gets a higher priority when it initiates route discovery.

SRP authenticates ROUTE REPLYS from intermediate nodes using shared group keys or digital signatures. When a node with a cached route shares a group key with (or can generate a digital signature verifiable by) the initiator of the REQUEST, it can use that group key to authenticate the REPLYS. The authenticator, which is either a message authentication code computed using the group key or a signature, is called the intermediate node reply token. The signature or MAC is computed over the cache REPLY.

As we mentioned earlier, SRP doesn't attempt to address the route-maintenance question. In SRP, as in Ariadne, multiple REPLYS are returned for each REQUEST; nodes use secure message transmission (SMT)[26] to ensure successful delivery of data packets. In SMT, data messages are split into packets using secret sharing techniques so that if M out of N such packets are received, the message can be reconstructed.

Securing AODV

The Ad hoc On-demand Distance Vector routing protocol (AODV) spreads distance vector routing information in an on-demand manner. Researchers have designed two protocols to secure routing protocols based on this design.

Authenticated routing for ad hoc networks

Kimaya Sanzgiri and her colleagues[27] developed authenticated routing for ad hoc networks (ARAN), which is based on AODV. In ARAN, each node has a certificate signed by a trusted authority, which associates its IP address with a public key. ARAN is an on-demand protocol, broken up into route discovery and maintenance.

Figure 2 shows an example of route discovery in ARAN. To initiate a route discovery, the initiator (in this example, S) broadcasts a signed ROUTE REQUEST packet that includes the target (D in the example), its certificate (cert_S), a nonce N, and a timestamp t. The nonce and timestamp together ensure freshness when used in a network with a limited clock skew. Each node that forwards this REQUEST checks the signature or signatures. In our example, node C checks node B's certificate cert_B, then checks the signature on the outer message. C then verifies

the certificate $cert_S$ for initiator S and uses the key in the certificate to verify the signature on the `REQUEST`. If the signatures (or signature, when the packet is directly received from the initiator) are valid, the forwarding node removes the last forwarder's signature and certificate (if applicable), signs the original `REQUEST`, and includes its own certificate. The node then broadcasts the `REQUEST`. In the example, node C removes node B's signature, signs the resulting `REQUEST`, and includes its own certificate. Node C then broadcasts the `REQUEST`.

When the first `ROUTE REQUEST` from a route discovery reaches the target, the target signs a `ROUTE REPLY` and sends it to the node from which it received the `REQUEST`. In our example, the target D returns a signed `ROUTE REPLY` to the previous hop C. The `ROUTE REPLY` is forwarded in much the same way as the `REQUEST`, except that each node unicasts the `REPLY` to the node from which it received the `REQUEST`. In particular, each node receiving a `REPLY` checks the signature or signatures. In our example, node B checks node C's certificate $cert_C$, then checks the signature on the outer message. B then verifies target D's certificate $cert_D$ and uses the key in the certificate to verify the signature on the `REQUEST`. If the signatures (or signature, when the packet is directly received from the target) are valid, the forwarding node removes the last forwarder's signature and certificate (if applicable), signs the original `REPLY`, and includes its own certificate. It then unicasts the `REPLY` to the node from which it received the associated `REQUEST`. In the example, node B removes node C's signature, signs the resulting `REPLY`, and includes its own certificate. Node B then unicasts the resulting `REPLY` to A, from which it had previously heard the `REQUEST`.

When a node B forwards a `REPLY` packet that it received from node C toward the previous node A, it also establishes a routing table entry for the target D, indicating that the next-hop destination for packets destined toward D is node C. When packets destined for destination D are forwarded to node B, it will in turn forward them to node C. If node B discovers that the link from itself to node C is broken, and thus it can't forward packets to node C, it initiates route maintenance. Figure 3 shows an example of ARAN route maintenance. The intermediate node sends a `ROUTE ERROR` to the previous hop, indicating that the route has been broken. This `ROUTE ERROR` includes the source, destination, intermediate node certificate, and a nonce and timestamp generated by the intermediate node for freshness. This packet is forwarded unchanged to the source.

Because ARAN uses public-key cryptography for authentication, it is particularly vulnerable to DoS attacks based on flooding the network with bogus control packets for which signature verifications are required. As long as a node can't verify signatures at line speed, an attacker can force that node to discard some fraction of the control packets it receives.

$B \rightarrow A$: $\langle$(`ROUTE ERROR`, S, D, $cert_B$, N, t)$_{K_B^-}\rangle$
$A \rightarrow S$: $\langle$(`ROUTE ERROR`, S, D, $cert_B$, N, t)$_{K_B^-}\rangle$

Figure 3. Route maintenance in ARAN. When a node B determines that its next-hop to D is unreachable, it broadcasts a signed route error message indicating that its next hop to D is unreachable. Each node using B as a next-hop for D rebroadcasts this route error but does not re-sign it.

SAODV

Manel Guerrero Zapata and N. Asokan[28] propose Secure AODV (SAODV), another protocol designed to secure AODV. The idea behind SAODV is to use a signature to authenticate most fields of a route request (RREQ) and route reply (RREP) and to use hash chains to authenticate the hop count. SAODV designs signature extensions to AODV. Network nodes authenticate AODV routing packets with an SAODV signature extension, which prevents certain impersonation attacks. In SAODV, an RREQ packet includes a route request single signature extension (RREQ-SSE). The initiator chooses a maximum hop count, based on the expected network diameter, and generates a one-way hash chain of length equal to the maximum hop count plus one. This one-way hash chain is used as a metric authenticator, much like the hash chain within SEAD. The initiator signs the RREQ and the anchor of this hash chain; both this signature and the anchor are included in the RREQ-SSE. In addition, the RREQ-SSE includes an element of the hash chain based on the actual hop count in the RREQ header. We call this value the hop-count authenticator. For example, if the hash chain values h_0, h_1, ..., h_N were generated such that $h_i = H[h_{i+1}]$, then the hop-count authenticator h_i corresponds to a hop count of $N - i$.

With the exception of the hop-count field and hop-count authenticator, the fields of the RREQ and RREQ-SSE headers are immutable and therefore can be authenticated by verifying the signature in the RREQ-SSE extension. To verify the hop-count field in the RREQ header, a node can follow the hash chain to the anchor. For example, if the hop-count field is i, then hop-count authenticator hca should be $H^i[h_N]$. Because the length (N) and anchor (h_N) of this hash chain is included in the RREQ-SSE and authenticated by the signature, a node can follow the hash chain and ensure that $h_N = H^{N-i}[hca]$. Figure 4 shows an example of route discovery in SAODV.

When forwarding an RREQ in SAODV, a node first authenticates the RREQ to ensure that each field is valid. It then performs duplicate suppression to ensure that it forwards only a single RREQ for each route discovery. The node then increments the hop-count field in the RREQ header, hashes the hop count authenticator, and rebroadcasts the RREQ, together with its RREQ-SSE extension.

$S \rightarrow *$: $\langle$(RREQ, id, S, seq$_S$, D, oldseq$_D$, h$_0$, N) $_{K_S^-}$, 0, $h_N\rangle$
$A \rightarrow *$: $\langle$(RREQ, id, S, seq$_S$, D, oldseq$_D$, h$_0$, N) $_{K_S^-}$, 1, $h_{N-1}\rangle$
$B \rightarrow *$: $\langle$(RREQ, id, S, seq$_S$, D, oldseq$_D$, h$_0$, N) $_{K_S^-}$, 2, $h_{N-2}\rangle$
$C \rightarrow *$: $\langle$(RREQ, id, S, seq$_S$, D, oldseq$_D$, h$_0$, N) $_{K_S^-}$, 3, $h_{N-3}\rangle$
$D \rightarrow C$: $\langle$(RREP, D, seq$_D$, S, lifetime, h'_0, N) $_{K_D^-}$, 0, $h'_N\rangle$
$C \rightarrow B$: $\langle$(RREP, D, seq$_D$, S, lifetime, h'_0, N) $_{K_D^-}$, 1, $h'_{N-1}\rangle$
$B \rightarrow A$: $\langle$(RREP, D, seq$_D$, S, lifetime, h'_0, N) $_{K_D^-}$, 2,$h'_{N-2}\rangle$
$A \rightarrow S$: $\langle$(RREP, D, seq$_D$, S, lifetime, h'_0, N) $_{K_D^-}$, 3, $h'_{N-3}\rangle$

Figure 4. Route discovery in SAODV. In this figure, node S is discovering a route to node D. The main differences between ARAN and SAODV route discovery are that ARAN uses an extra signature to authenticate the previous hop, and SAODV uses a hash chain to authenticate the metric (as in SEAD).

$B \rightarrow A$: (RERR, D, seq$_D$) $_{K_B^-}$
$A \rightarrow S$: (RERR, D, seq$_D$) $_{K_A^-}$

Figure 5. Route maintenance in SAODV. The main differences between ARAN and SAODV route maintenance is that each SAODV that forwards a route error signs it, whereas forwarding nodes in ARAN simply rebroadcast the packet.

When the RREQ reaches the target, the target checks the authentication in the RREQ-SSE. If the RREQ is valid, the target returns an RREP as in AODV. A route reply single signature extension (RREP-SSE) provides authentication for the RREP. As in the RREQ, the only mutable field is the hop count; as a result, the RREP is secured in the same way as the RREQ. In particular, an RREP-SSE has a signature covering the hash chain anchor together with all RREP fields except the hop count. The hop count is authenticated by a hop-count authenticator, which is also a hash chain element. As before, a hop-count authenticator of h_i corresponds to a hop count of $N - i$.

A node forwarding an RREP checks the signature extension. If the signature is valid, then the forwarding node sets its routing table entry for the RREP's original source, specifying that packets to that destination should be forwarded to the node from which the forwarding node heard the RREP. For example, in Figure 4, when node B forwards the RREP from C, it sets its next hop for destination D to C.

SAODV allows intermediate-node replies through the use of a route reply double signature extension (RREP-DSE). An intermediate node replying to an RREQ includes an RREP-DSE. The idea here is that to establish a route to the destination, an intermediate node must have previously forwarded an RREP from the destination. If the intermediate node had stored the RREP and signature, it can then return the same RREP if the sequence number in that RREP is greater than the sequence number specified in the RREQ. However, some of the fields of that RREP, in particular the lifetime field, are no longer valid. As a result, a second signature, computed by the intermediate node, is used to authenticate this field.

To allow replies based on routing information from an RREQ packet, the initiator includes a signature suitable for an RREP packet through the use of an RREQ-DSE. Conceptually, the RREQ-DSE is an RREQ and RREP rolled into one packet. To reduce overhead, SAODV uses the observation that the RREQ and RREP fields substantially overlap. In particular, the RREQ-DSE need only include some flags, a prefix size, and some reserved fields, together with a signature valid for an RREP using those values. When a node forwards an RREQ-DSE, it caches the route and signature in the same way as if it had forwarded an RREP.

SAODV also uses signatures to protect the route error (RERR) message used in route maintenance. In SAODV, each node signs the RERR it transmits, whether it's originating the RERR or forwarding it. Nodes implementing SAODV don't change their destination sequence number information when receiving an RERR because the destination doesn't authenticate the destination sequence number. Figure 5 shows an example of SAODV route maintenance.

Under attack, ARAN need only verify one signature in an attacker's packet by blacklisting a node that doesn't correctly verify the inside signature—the initiator's signature in the case of an RREQ or the target's signature in the case of an RREP. An attacker, then, is unlikely to include a valid outer signature with an invalid inner signature. As a result, any bogus packet would have only a bogus outer signature and, hence, have the same verification cost as SAODV.

Securing link-state routing

Panagiotis Papadimitratos and Haas[29] also propose the Secure Link-State Protocol (SLSP), which uses digital signatures and one-way hash chains to ensure the security of link-state updates. We can use SLSP as the Intrazone Routing Protocol in the Zone Routing Protocol (ZRP). SLSP is a periodic protocol that receives link-state information through a periodic Neighbor Location Protocol. As a part of NLP, each node broadcasts a signed pairing between its IP address and its MAC address. A node's NLP can notify SLSP when one MAC address uses two IP addresses, when two MAC addresses claim the same IP address, and when another node uses the same MAC address as the detecting node. These protocols ensure some level of integrity of MAC and IP addresses within a two-hop radius.

SLSP link-state updates are signed and propagated a limited number of hops. In ZRP, SLSP link-state updates

would have a maximum hop count equal to a zone radius. To ensure that an SLSP update doesn't travel too many hops, each update includes a hop count representing the number of hops traveled by the SLSP update. As in SEAD and SAODV, a hash chain is used to authenticate the hop count, and the hash chain values are authenticated using the hash chain's anchor, which is included in the signed portion of the SLSP link-state update.

SLSP uses the same lightweight flooding prevention mechanism as SRP, wherein nodes that relay or generate fewer link-state updates are given priority over any node that sends more link-state updates. As in SRP, an attacker can masquerade as a victim node and flood the victim's neighbors with link-state updates that appear to originate at the victim. Although the victim might be able to detect the attack, due to NLP's duplicate MAC address detection functionality, the victim will have no way to protest.

Reputation-based systems

Confidant,[30] based on DSR, consists of four components: the monitor, the trust monitor, the reputation system, and the path manager. For each packet a node forwards, the monitor on that node attempts to ensure that the next-hop node also forwarded the packet correctly. When the monitor detects an anomaly, it triggers action by the reputation system, which maintains a local ratings list. These lists are potentially exchanged with other nodes; the trust monitor handles input from other nodes. If a list is received from a highly trusted node, the receiver can directly place information from the list into its local ratings list. On the other hand, if a list is received from an untrusted source, the receiver can completely ignore it or give it substantially less weight than a list received from a more trusted node. Finally, the path manager chooses paths from the node's route cache based on a blacklist and the local ratings list. The path manager also specifies the reaction to a `REQUEST` from a node on the blacklist or to a `REQUEST` that has traversed a node on the blacklist.

Baruch Awerbush and his colleagues[31] propose a secure on-demand ad hoc network routing protocol. In this protocol, routes are discovered on-demand, as in DSR. Each `ROUTE REQUEST` packet includes the same fields used in DSR, except they include a weight list instead of a node list. This weight list is a list of links, together with the cost (metric) associated with each such link; these costs are derived from the initiator's previous experience sending packets over that link. When the flooded `REQUEST` reaches the target, it floods a `REPLY`. Each node forwards a `REPLY` if it has not previously forwarded a better reply. In particular, each node computes the total cost of the recorded hops in a `REPLY` and adds the cost of the hop between the previous node and itself. If this total cost is less than the total cost of the previously forwarded `REPLY` (to this `REQUEST`), then the node also forwards this `REPLY`.

Relative to Ariadne, this approach provides additional flexibility in specifying which routes to prefer because Ariadne returns a set of node-disjoint, lowest-latency paths subject to a blacklist specified in the `REQUEST`. Unlike Ariadne, their technique is vulnerable to the hop-drop attack, where a node forwarding a `REPLY` removes one or more nodes from the `REPLY` that it forwards. Furthermore, the number of `REPLY` packets sent in response to a single `REQUEST` is potentially exponential, even in the absence of an attacker. For example, suppose there are n nodes in addition to the initiator and the target, arranged in $n/2$ groups of two nodes. Let group 0 represent the target and group $n/2 + 1$ represent the initiator. If these groups are arranged so that both nodes in group i are neighbors of all nodes in groups $i - 1$, i, and $i + 1$, and of no other nodes, then group 1 would forward two `REPLYs`. If the higher-metric node in group 1 was the first to forward the `REPLY`, then each node in group 2 would forward both `REPLYs`, for a total of four `REPLYs` forwarded. Again, in the worst case, each node in group 3 would forward all four `REPLYs`. In general, group i would forward 2^i `REPLY` packets, for a total of

$$\sum_{i=0}^{n/2} 2^i = 2^{\frac{n}{2}+1} - 1$$

`REPLY` packet transmissions.

Awerbush and his colleagues also propose a technique for performing route maintenance in cases where an attacker is already on the path. Their approach is to define an acceptable level of performance—for example, based on a packet delivery ratio within a latency limit. When a path's performance drops below the acceptable level, a binary search is initiated to locate the link responsible for dropping the path's performance level below the acceptable level. A digital signature authenticates these detection packets are authenticated, which are then "onion" encrypted, such that each node forwarding the packet decrypts the outer layer, processes any probe requests, and forwards the packet.

This binary search technique works well in a path with only a single attacker, when the attacker causes the vast majority of packet losses on the path. However, an attacker can still cause the protocol to detect another link as the attacking link, simply by dropping several packets slightly less than the threshold. It is likely that subsequent nodes will occasionally drop a packet; in such cases, any binary search would show that performance up to and including the attacking node is acceptable but might find a fault further down the path.

Secure Traceroute[32] uses a similar detection scheme. It differs in that probe packets are indistinguishable from ordinary packets, and it can use secret sharing to differentiate between individual losses and losses beyond a certain threshold. Another difference is that Secure Traceroute

performs a linear search, as opposed to a binary search. Secure Traceroute is therefore more robust against multiple attackers on the route, but is slower to find a faulty link. Another approach to avoiding malicious nodes is choosing routes based on reinforcement learning.[31] In this approach, each node uses its previous experience with various links to choose a path.

Reputation-based systems rely on authentic information in routing headers to correctly attribute malicious behavior. As a result, they require an underlying secure ad hoc network routing protocol. When used with such a protocol, they can provide a powerful deterrent to malicious behavior by providing the threat of blacklisting. When such systems are used, the attacker's game-theoretic optimal behavior is likely to be substantially muted, thus improving network performance. Designers of a reputation-based system, however, must avoid the possibility of a malign attack, where an attacker causes legitimate nodes to be blacklisted.

Open research challenges

A number of challenges remain in the area of securing wireless ad hoc networks. First, the secure routing problem in such networks isn't well modeled. A more complete model of possible attacks would let protocol designers evaluate the security of their routing protocol. In addition, such a model would form the basis for using formal methods to verify protocol security.

Another problem is designing efficient routing protocols that have both strong security and high network performance. Although researchers have designed security extensions for several existing protocols, many of these extensions remove important performance optimizations. Optimistic approaches can provide a better tradeoff between security and performance. □

References

1. S. Marti et al., "Mitigating Routing Misbehaviour in Mobile Ad Hoc Networks," *Proc. 6th Ann. Int'l Conf. Mobile Computing and Networking* (MobiCom 2000), ACM Press, 2000, pp. 255–265.
2. Y.-C. Hu, A. Perrig, and D.B. Johnson, "Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols," *Proc. 2003 ACM Workshop on Wireless Security* (WiSe 2003), ACM Press, 2003, pp. 30–40.
3. Y.-C. Hu, A. Perrig, and D.B. Johnson, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks," *Proc. 22nd Ann. Joint Conf. IEEE Computer and Communications Societies* (INFOCOM 2003), IEEE Press, 2003, pp. 1976–1986.
4. D.B. Johnson, "Routing in Ad Hoc Networks of Mobile Hosts," *Proc. IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA'94), IEEE Press, 1994, pp. 158–163.
5. C.E. Perkins and E.M. Royer, "Ad-Hoc On-Demand Distance Vector Routing," *Proc. 2nd IEEE Workshop Mobile Computing Systems and Applications* (WMCSA'99), IEEE Press, 1999, pp. 90–100.
6. A. Qayyum, L. Viennot, and A. Laouiti, "*Multi-Point Relaying: An Efficient Technique for Flooding in Mobile Wireless Networks*," tech. report RR-3898, INRIA, Feb. 2000.
7. B. Bellur and R.G. Ogier. "A Reliable, Efficient Topology Broadcast Protocol for Dynamic Networks," *Proc. 18th Ann. Joint Conf. IEEE Computer and Communications Societies* (INFOCOM '99), IEEE Press, 1999, pp. 178–186.
8. M. Corner and B. Noble, "Zero-Interaction Authentication," *Proc. 8th ACM Int'l Conf. Mobile Computing and Networking* (MobiCom 2002), ACM Press, 2002, pp. 1–11.
9. T. Kindberg, K. Zhang, and N. Shankar, "Context Authentication Using Constrained Channels," *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA 2002), IEEE Press, 2002, pp. 14–21.
10. A. Perrig et al., "Efficient Authentication and Signing of Multicast Streams over Lossy Channels," *Proc. IEEE Symp. Security and Privacy*, IEEE Press, 2000, pp. 56–73.
11. R.C. Merkle, "Protocols for Public Key Cryptosystems," *Proc. IEEE Symp. Research in Security and Privacy*, IEEE Press, 1980, pp. 122–133.
12. Y.-C. Hu, A. Perrig, and D.B. Johnson, "Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks," *Proc. 8th Ann. Int'l Conf. Mobile Computing and Networking* (MobiCom 2002), ACM Press, 2002, pp. 12–23.
13. F. Stajano and R. Anderson. "The Resurrecting Duckling: Security Issues for Ad-Hoc Wireless Networks," *Proc. 7th Int'l Workshop on Security Protocols*, Lecture Notes in Computer Science 1796, Springer-Verlag, 1999, pp. 172–194.
14. D. Balfanz et al., "Talking to Strangers: Authentication in Ad-Hoc Wireless Networks," *Proc. Symp. Network and Distributed Systems Security* (NDSS 2002), Internet Society, 2002, pp. 23–35.
15. W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Trans. Information Theory*, vol. 22, Nov. 1976, pp. 644–654.
16. G. Montenegro and C. Castelluccia. "Statistically Unique and Cryptographically Verifiable (SUCV) Identifiers and Addresses," *Proc. Symp. Network and Distributed Systems Security* (NDSS 2002), Internet Society, 2002, pp. 87–99.
17. L. Zhou and Z.J. Haas, "Securing Ad Hoc Networks," *IEEE Network Magazine*, IEEE Press, vol. 13, no. 6, 1999, pp. 24–30.
18. S. Yi and R. Kravets, *Key Management for Heterogeneous Ad Hoc Wireless Networks*, tech. report UIUCDCS-R-2002-2290, Dept. Computer Science, Univ. of Illinois at Urbana-Champaign, July 2002.
19. J.-P. Hubaux, L. Buttyán, and S. Čapkun, "The Quest for Security in Mobile Ad Hoc Networks," *Proc. 2nd Symp. Mobile Ad Hoc Networking and Computing* (MobiHoc

2001), ACM Press, 2001, pp. 146–155.
20. Y.-C. Hu, D.B. Johnson, and A. Perrig, "SEAD: Secure Efficient Distance Vector Routing in Mobile Wireless Ad Hoc Networks," *Proc. 4rth IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA 02), IEEE Press, 2002, pp. 3–13.
21. C.E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," *Proc. SIGCOMM '94 Conf. Communications Architectures, Protocols and Applications*, ACM Press, 1994, pp. 234–244.
22. M. Mathis et al., *TCP Selective Acknowledgment Options*," RFC 2018, Oct. 1996.
23. T. Narten, E. Nordmark, and W. Allen Simpson, "*Neighbor Discovery for IP Version 6* (IPv6)," RFC 2461, Dec. 1998.
24. P. Papadimitratos and Z.J. Haas, "Secure Routing for Mobile Ad Hoc Networks," *Proc. SCS Communication Networks and Distributed Systems Modeling and Simulation Conf.* (CNDS 2002), Jan. 2002.
25. Z.J. Haas and M.R. Pearlman, "The Performance of Query Control Schemes for the Zone Routing Protocol," *Proc. ACM SIGCOMM 98 Conf. Applications, Technologies, Architectures, and Protocols for Computer Communication*, ACM Press, 1998, pp. 167–177.
26. P. Papadimitratos and Z.J. Haas, "Secure Message Transmission in Mobile Ad Hoc Networks," *Elsevier Ad Hoc Networks J.*, Elsevier, vol. 1, no. 1, 2003, pp. 193–209.
27. K. Sanzgiri et al., "A Secure Routing Protocol for Ad Hoc Networks," *Proc. 10th IEEE Int'l Conf. Network Protocols* (ICNP '02), IEEE Press, 2002, pp. 78–87.
28. M. Guerrero Zapata and N. Asokan, "Securing Ad Hoc Routing Protocols," *Proc. ACM Workshop on Wireless Security* (WiSe), ACM Press, 2002, pp. 1–10.
29. P. Papadimitratos and Z.J. Haas, "Secure Link State Routing for Mobile Ad Hoc Networks," *Proc. IEEE Workshop on Security and Assurance in Ad Hoc Networks*, IEEE Press, 2003, pp. 27–31.
30. S. Buchegger and J.-Y. Le Boudec, "Performance Analysis of the CONFIDANT Protocol (Cooperation Of Nodes - Fairness In Dynamic Ad-hoc NeTworks)," *Proc. 3rd Symp. Mobile Ad Hoc Networking and Computing* (MobiHoc 2002), ACM Press, 2002, pp. 226–236.
31. B. Awerbuch et al., "An On-Demand Secure Routing Protocol Resilient to Byzantine Failures," *ACM Workshop on Wireless Security* (WiSe), ACM Press, 2002, pp. 21–30.
32. V.N. Padmanabhan and D.R. Simon. "Secure Traceroute to Detect Faulty or Malicious Routing," *Proc. 1st Workshop on Hot Topics in Networks* (HotNets-I), 2002, pp. 77–82.

Yih-Chun Hu *is a postdoctoral research fellow at the University of California, Berkeley. In his thesis work at Carnegie Mellon, he focused on security and performance in wireless ad hoc networks. His research interests include mobility and security in wired and wireless networks. He is a member of the ACM. Contact him at yihchun@cs.cmu.edu.*

Adrian Perrig *is an assistant professor at Carnegie Mellon University in the Electrical and Computer Engineering, Engineering and Public Policy, and Computer Science departments. His research revolves around security, focusing on security for sensor networks, security for ad hoc networks, and security for the Internet. He is a member of ACM, IEEE, and Usenix. Contact him at adrian@ece.cmu.edu.*