

Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks

Yih-Chun Hu Adrian Perrig David B. Johnson
Rice University University of California, Berkeley Rice University
yihchun@cs.cmu.edu perrig@cs.berkeley.edu dbj@cs.rice.edu

Rice University
Department of Computer Science
Technical Report TR01-383

December 17, 2001
Revised: March 22, 2002

Abstract

An *ad hoc network* is a group of wireless mobile computers (or nodes), in which individual nodes cooperate by forwarding packets for each other to allow nodes to communicate beyond direct wireless transmission range. Prior research in ad hoc networking has generally studied the routing problem in a non-adversarial network setting, assuming a reasonably trusted environment. In this paper, we present attacks against routing in ad hoc networks, and we present the design and performance evaluation of a new secure on-demand ad hoc network routing protocol, called Ariadne, that prevents attackers (or compromised nodes) from tampering with uncompromised routes consisting of uncompromised nodes. Ariadne is efficient: it only uses highly efficient *symmetric* cryptographic primitives. Ariadne also prevents a large number of denial-of-service attacks.

1. Introduction

An *ad hoc network* is a group of mobile nodes using wireless network interfaces, in which individual nodes cooperate by forwarding packets for each other to allow nodes to communicate beyond direct transmission range. Ad hoc networks require no centralized administration or fixed network infrastructure such as base stations, and can be quickly and inexpensively set up as needed. Ad hoc networks can be used in scenarios where no infrastructure exists, or where the existing infrastructure does not meet application requirements for reasons such as security, cost, or quality. Applications such as military exercises, disaster relief, and mine site operation may benefit from ad hoc networking, but secure and reliable communication is a necessary prerequisite for such applications.

Ad hoc network routing protocols are challenging to design, and secure ones are even more so. Wired network routing protocols such as BGP [76] do not handle well the type of rapid node mobility and network topology changes that occur in ad hoc networks; such protocols also have high communication overhead because they send periodic routing messages even when the network is not changing. So far, researchers in ad hoc networking have generally studied the routing problem in a non-adversarial network setting, assuming a reasonably trusted environment; relatively little research has been done in a more realistic setting in which an adversary may attempt to disrupt the communication.

We focus here on *on-demand* (or reactive) routing protocols for ad hoc networks, in which a node attempts to discover a route to some destination only when it has a packet to send to that destination [22, 36, 42, 60, 63]. On-demand routing protocols have been demonstrated to perform better with significantly lower overheads than periodic (or proactive) routing protocols in many situations [12, 34, 48, 63], since they are able to react quickly to the many changes that may occur in node connectivity, yet are able to reduce (or eliminate) routing overhead in periods or areas of the network in which changes are less frequent.

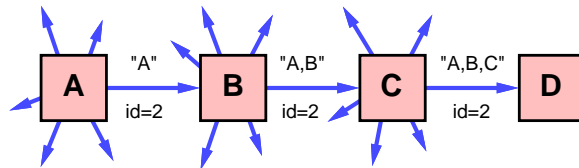


Figure 1: Example of DSR Route Discovery

In this paper, we make several contributions to the area of secure routing protocols for ad hoc networks:

- We give a model for the types of attacks possible in such a system, and we describe a number of novel attacks on ad hoc network routing protocols.
- We present the design and performance evaluation of a new on-demand secure ad hoc network routing protocol, called Ariadne, that withstands node compromise and relies only on highly efficient *symmetric* cryptography. Relative to previous work in securing ad hoc network routing protocols (e.g., [5, 19, 50, 59, 68, 88, 90]), Ariadne is either more secure, more efficient, or more general (e.g., Ariadne does not require a trusted infrastructure and does not require powerful processors).

In Section 2, we summarize the basic operation of the Dynamic Source Routing protocol (DSR) [35, 36, 37], on which we base the design of our new secure routing protocol, and in Section 3, we describe TESLA [66], on which we base the authentication mechanism of Ariadne. Section 4 describes our assumptions about the network, the nodes, and security and key setup. We present an attacker model and describe types of attacks in Section 5; and in Section 6, we present the design of our new secure ad hoc network routing protocol, Ariadne. Section 7 gives an initial simulation-based performance evaluation of a basic form of Ariadne. In Section 8, we discuss related work, and we present our conclusions in Section 9.

2. Basic Operation of DSR

We base the design of our secure on-demand ad hoc network routing protocol, Ariadne, on the basic operation of the Dynamic Source Routing protocol (DSR) [35, 36, 37], since DSR operates *entirely* on-demand and has been well studied through both simulation and real testbed implementation [12, 34, 48, 49]. Unlike periodic protocols (e.g., [7, 62, 72]), which exchange routing information between nodes periodically in an attempt to always maintain routes to all destinations, on-demand protocols exchange routing information only when a packet needs to be delivered to some destination. On-demand approaches to routing in ad hoc networks often have lower overhead than periodic protocols, since they transmit routing information only in response to actual packets to be sent or in response to topology changes affecting routes actively in use. Lower routing overhead allows more of the available bandwidth and battery power to be used towards delivery of application data. In a secure routing protocol, reduced overhead has the added benefit of reducing the number of routing packets that need to be authenticated, thereby reducing the computational overhead needed for security.

DSR divides the routing problem into two parts: *Route Discovery* and *Route Maintenance*. In this section, we describe the basic form of Route Discovery and Route Maintenance in DSR. The DSR protocol also defines a number of optimizations to these mechanisms [30, 35, 36, 37, 42], but the use of these optimizations is beyond the scope of this paper. In this paper, we thus secure only a basic version of DSR.

In DSR, when a node has a packet to send to some destination and does not currently have a route to that destination in its *Route Cache*, the node initiates a Route Discovery to find a route, as illustrated in Figure 1. The initiator of the Route Discovery (node *A*) broadcasts a ROUTE REQUEST packet, specifying the target of that Discovery (node *D*) and a unique identifier from *A*. Each node receiving the ROUTE REQUEST, if it has recently seen this request identifier from the initiator, discards the REQUEST. Otherwise, it appends its own node address to a list in the REQUEST and rebroadcasts the REQUEST. When the ROUTE REQUEST reaches its target node, the target sends a ROUTE REPLY back to the initiator of the REQUEST, including a copy of the list of addresses from the REQUEST. When the REPLY reaches the initiator of the REQUEST, it caches the new route in its Route Cache.

Route Maintenance is the mechanism by which a node sending a packet along a specified route to some destination detects if that route has broken, for example because two nodes in it have moved too far apart. DSR uses *source routing*: when sending a packet, the originator includes in the header of the packet the complete sequence of nodes through which the packet is to be forwarded. Each node along the route forwards the packet to the next hop indicated in the packet's header, and attempts to confirm that the packet was received by that next node; a node may confirm

this by means of a link-layer acknowledgment (such as that provided by the IEEE 802.11 MAC protocol [33]), by means of a passive acknowledgment [38], or by a network-layer acknowledgment. If, after a limited number of local retransmissions of the packet, a node in the route (e.g., node B) is unable to make this confirmation, it returns a ROUTE ERROR to the original source of the packet (node A), identifying the broken link from itself to the next node (e.g., the link from B to C). The sender then removes the identified broken link from its Route Cache; for subsequent packets to this destination, the sender may use any other route to that destination in its Cache, or it may attempt a new Route Discovery for that target if necessary.

3. Overview of TESLA

We base the authentication in Ariadne on the TESLA broadcast authentication protocol [65, 66], since TESLA is efficient and adds only a single message authentication code (MAC) to a message for authentication¹. Adding a MAC (computed with a shared key) to a message provides secure authentication in point-to-point communication; in broadcast however, multiple receivers need to know the MAC key for verification, which would also allow any receiver to forge packets and impersonate the sender. Secure broadcast authentication thus requires an asymmetric primitive, such that the sender can generate valid authentication information, but that the receivers can only verify the authentication information. Unlike traditional asymmetric protocols such as RSA [78], which create asymmetry using computationally expensive one-way trapdoor functions, TESLA uses clock synchronization and delayed key disclosure to create the asymmetry for secure broadcast authentication from *symmetric* primitives. In this section, we summarize a simplified version of TESLA.

TESLA uses one-way key chains. One-way key chains are a widely used cryptographic primitive: One of the first uses of one-way chains was for one-time passwords by Lamport [46]. Haller later used the same approach for the S/KEY one-time password system [23]. One-way chains are also used in efficient one-time signature algorithms [20, 53, 54, 79], (micro-)payment mechanisms [2, 25, 61, 77], server-supported non-repudiation [3], and for conditional anonymity [26]. Hash chains were also used for efficient certificate revocation [55], and to authenticate link-state routing updates [15, 24, 89].

Each sender using TESLA for authentication generates a *one-way key chain*, by repeatedly computing a one-way hash function H on a randomly chosen key K_N . For example, $K_{N-1} = H[K_N]$, $K_{N-2} = H[K_{N-1}]$. More generally, $K_i = H^{N-i}[K_N]$. This one-way chain has two main properties. First, anybody can compute the key chain in one direction, hence anybody can derive previous keys K_j from key K_i , where $j < i$. Second, any key can be used to authenticate following keys. For example, if a receiver knows the authentic key K_i , it can authenticate the following key K_j by verifying $K_i = H^{j-i}[K_j]$.

The sender pre-determines a schedule at which it publishes (or discloses) the keys of the one-way key chain in reverse order from generation, i.e., the sender publishes K_0, K_1, \dots, K_N . For example, a simple key disclosure schedule would be to publish key K_i at time $T_0 + i \cdot t$, where T_0 is the time at which K_0 is published, and t is the duration between published keys.

Central to TESLA's security is a receiver's ability to determine which keys a sender may have already published. A receiver using TESLA relies on loose time synchronization, together with a known maximum time synchronization error between any two nodes Δ , to determine the maximum time at the sender. If a sender discloses its first key K_0 at time T_0 (on its clock) and discloses an additional key each time interval, then the receiver can determine, based on its clock, Δ , and T_0 , whether or not some key K_i has yet been disclosed.

To send a packet, the sender first estimates a pessimistic upper bound on the end-to-end network delay. The sender then picks a key K_i from its one-way key chain which the receiver will believe is still secret at the time the receiver is expected to receive the packet. For example, if the sender believes the end-to-end delay would at worst² be τ , it could choose a key K_i that would not be disclosed until a time at least $\tau + 2\Delta$ in the future.³ The sender adds a message authentication code (MAC), computed using key K_i , to the packet, and sends it to the receiver.

When a receiver receives a packet authenticated with TESLA, it first verifies that the key K_i used to authenticate that packet is still secret. For example, given the packet arrival time t' and the maximum time synchronization error

¹The term "MAC" in this paper always refers to message authentication code; we will always spell out Medium Access Control to avoid confusion.

²Even if the end-to-end delay is larger than τ , TESLA is still secure, but some receivers will need to discard the packet because the sender may have already disclosed the key, as we describe below.

³The value 2Δ is used here because the receiver's clock may be ahead of the sender's clock by Δ , so at time t_s at the sender, it is $t_s + \Delta$ at the receiver. When the packet reaches the receiver, it will be $t_s + \tau + \Delta$, and the receiver will discard the packet if the key *might* have been released; since the sender's clock may be Δ ahead of the receiver, the receiver will reject the packet if the key was to be released at time $t_s + \tau + 2\Delta$ or earlier.

Δ , the receiver checks that the time elapsed between time T_0 and $t' - \Delta$ does not exceed i time intervals. If the check fails, the sender may have already published K_i and an attacker may have forged the packet contents; the receiver thus discards the packet. However, if the check is successful, the receiver buffers the packet and waits for the sender to publish key K_i . When the receiver receives K_i , it first authenticates K_i , and then authenticates stored packets authenticated with a key K_j , where $j \leq i$.

4. Assumptions

4.1. Network Assumptions

Wireless physical layers for sending data from one node to another are often vulnerable to denial of service attacks such as jamming. Mechanisms such as spread spectrum [69] have been extensively studied as means of providing resistance to physical jamming, and we thus disregard such attacks here.

We assume that network links are bidirectional; that is, if a node A is in transmission range of some node B , then B is in transmission range of A . When nodes use equal power levels and identical coding for all link-layer frames (packets) transmitted, and when all nodes use omnidirectional antennas, links are generally always bidirectional. Furthermore, many wireless Medium Access Control protocols require bidirectional links, as they exchange of several link-layer frames between a source and destination to help avoid collisions [10, 33].

Medium Access Control protocols are also often vulnerable to attack. For example, in IEEE 802.11, an attacker can paralyze nodes in its neighborhood by sending Clear-To-Send (CTS) frames periodically, setting the “Duration” field of each frame equal to the interval between such frames. Less sophisticated Medium Access Control protocols, such as ALOHA and Slotted ALOHA [1], are not vulnerable to such attacks but have lower efficiency. In this paper, we disregard attacks on Medium Access Control protocols.

We assume that the wireless network may drop, corrupt, reorder, or duplicate packets in transmission, for example due to interference effects or packet retransmissions. We further assume that the network includes a mechanism for transmission error detection in packets, such as a checksum or CRC code, and that nodes discard received packets for which this mechanism indicates that the packet has been corrupted. This mechanism, however, is not intended to replace cryptographic integrity checks on received packets.

To allow nodes to choose appropriate TESLA time intervals, as described in Section 3, we assume that each node in the network can estimate the end-to-end transmission time to any other node in the network. This value can be chosen adaptively, and can represent a pessimistic estimate. When this time is chosen to be too large, protocol responsiveness is reduced; when it is chosen to be too small, authentic packets may be rejected as forged, but forged packets are not accepted.

4.2. Node Assumptions

The resources of different ad hoc network nodes may vary greatly, from nodes with very minimal computational resources, such as Smart Dust [39], to resource-rich nodes perhaps equivalent in functionality to high-performance workstations. To make our results as general as possible, we design our protocol for nodes with minimal resources.

Most previous work on secure ad hoc network routing relies on *asymmetric* cryptography such as digital signatures [88, 90]. However, computing such signatures on resource-constrained nodes is quite expensive [5, 68], and we assume that some nodes in the ad hoc network may be so constrained. For example, Brown et al. analyze the computation time of digital signature algorithms on various platforms [13]: on a Palm Pilot or RIM pager, 163-bit Elliptic Curve Cryptography (ECC) [52] signature algorithms require 1.0–2.2 seconds of computation for one signature generation and 1.8–5.4 seconds for verification; a 512-bit RSA [78] signature requires 2.4–5.8 seconds of computation for generation and 0.1–0.6 seconds for verification, depending on the public exponent. Some other approaches provide efficient on-line signature generation but assume off-line preparation [20, 80, 81]. More recent signature algorithms provide short or efficient signatures, but their verification is slower than verification in RSA [18, 28, 70, 71].

We assume that all nodes in the ad hoc network have synchronized clocks within a maximum bound between any two nodes’ clocks of Δ . The value of the parameter Δ must be known by all nodes in the network. Though this time synchronization can be maintained with off-the-shelf hardware based on LORAN-C [56], WWVB [57], or GPS [17, 85], it is currently not a common part of ad hoc network nodes, and the time synchronization signal itself may be subject to attack [21]. Stable time sources such as microcomputer-compensated crystal oscillators [8] can provide sub-second accuracy for several months. Synchronization to within a few seconds per month can also be achieved

using temperature-compensated crystal oscillators. Finally, if normal crystal oscillators are used, Δ can be chosen to be as large as necessary, though a corresponding reduction in protocol responsiveness will result.

4.3. Security Assumptions and Key Setup

We assume some mechanism to bootstrap authentic keys required by our protocol. In particular, each node needs a shared secret key with each node it communicates with at a higher layer, an authentic TESLA key for each node in the ad hoc network, and an authentic “Route Discovery chain” element for each node for which this node will forward ROUTE REQUESTS. Section 6.7 presents a protocol that bootstraps authenticated keys with a KDC, given only shared secret keys with the KDC and an authentic TESLA key and “Route Discovery chain” element for the KDC, and does not rely on existing routing state in the ad hoc network. For example, a traditional approach is to use a Public-Key Infrastructure (PKI): each node has a public key certificate signed by a trusted Certificate Authority (CA), and every node has an authentic certificate of the CA’s public key. Another approach is to pre-load each node with symmetric keys for all other nodes before deployment of the ad hoc network. This approach does not require expensive asymmetric cryptographic operations for key setup and may be simple, for example, if the nodes of the ad hoc network all belong to a common administrative entity; however, the memory requirements may be large: this approach requires $N - 1$ pre-loaded keys per node (for networks with N nodes), for a total of $N(N - 1)/2$ keys. In some ad hoc networks, another possible approach for bootstrapping keys is the use of a trusted Key Distribution Center (KDC); the KDC may be either fixed or mobile. If each node shares a secret key with the KDC, two nodes can use the KDC to establish a shared secret key between them using a protocol such as Kerberos [43]; the KDC can also authenticate keys for other nodes. Stajano and Anderson also discuss the issues of bootstrapping keys in mobile devices [84].

5. Security of Ad Hoc Network Routing

In this section, we formalize an attacker model and discuss specific attacks against ad hoc network routing. This approach allows us to categorize the security of an ad hoc network routing protocol based on the strongest attacker it withstands.

5.1. Attacker Model

We consider the following possible configurations of attackers, with increasing strength:

- **Passive:** Passive attacker, only eavesdrops on the network.
- **Active1:** One active attacker node, has no cryptographic keys, but can inject packets into the network.
- **ActiveX:** Multiple active attacker nodes.
- **ActiveC:** One active attacker node having all cryptographic keys of one compromised node.
- **ActiveCX:** Multiple active attacker nodes having all cryptographic keys of one compromised node.
- **ActiveCCX:** Multiple active attacker nodes having all cryptographic keys of multiple compromised nodes.
- **ActiveVC:** All nodes on a vertex cut of the network have been compromised (attacker nodes have compromised all nodes on the vertex cut and know the cryptographic keys of these nodes). In this configuration, all packets being routed from one side of the network (on one side of the vertex cut) to the other must be routed through one of the compromised nodes.

Our protocol does not require a trusted KDC in the network, but some ad hoc networks may use one for key setup, as mentioned in Section 4.3. We do not consider the case in which an attacker compromises the KDC, since the KDC is a central trust entity, and a compromised KDC compromises the entire network.

5.2. General Attacks Against Ad Hoc Networks

Attacks on an ad hoc network generally fall into one of two categories: *routing disruption* attacks and *resource consumption* attacks. In a routing disruption attack, the attacker attempts to cause legitimate data packets to be routed in dysfunctional ways. In a resource consumption attack, 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, consuming energy and available bandwidth. An attacker may similarly create a routing *black hole* in which all packets are dropped; by sending forged routing packets, the attacker could route all packets for some destination to itself and then discard them, or the attacker could cause the route at all nodes in an area of the network to point “into” that area when in fact the destination is outside the area. As 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, forwarding routing packets but not data packets. An attacker may also attempt to cause a node to use *detours* (suboptimal routes) or may attempt to *partition* the network by injecting forged routing packets to prevent one set of nodes from reaching another. An attacker may attempt to make a route through itself appear longer by adding virtual nodes to the route; we call this attack *gratuitous detour*, as 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 watchdog and pathrater [50], an attacker may *blackmail* a good node, causing other good nodes to add that node to their blacklists, thus avoiding that node in routes.

A more subtle type of routing disruption attack is the creation of a *wormhole* in the network, using a pair of attacker nodes A and B linked via a private network connection. Every packet A receives from the ad hoc network, A forwards through the wormhole to B , to then be rebroadcast by B ; similarly, B may send all ad hoc network packets to A . Such an attack potentially disrupts routing by short circuiting the normal flow of routing packets, and the attackers may also create a virtual vertex cut that they control.

An example of a resource consumption attack is for an attacker to *inject extra data packets* into the network, which will consume bandwidth resources when forwarded, especially over detours or routing loops. Similarly, an attacker can *inject extra control packets* into the network, which may consume even more bandwidth or computational resources as other nodes process and forward such packets. With either of these attacks, an ActiveVC attacker can try to extract maximum resources from the nodes on both sides of the vertex cut, for example by forwarding only routing packets and not data packets, such that the nodes waste energy forwarding packets to the vertex cut, only to have them dropped.

If a routing protocol can prevent an attacker from inserting a loop in a route, and if a maximum route length can be enforced, then an attacker that can inject extra data packets has limited power. In particular, if routes are limited to ν hops, then each packet transmitted by the attacker only causes a fixed number of additional transmissions. More generally, if only one control packet is sent in response to each data packet, and that control packet is limited to ν hops, then an individual data packet can cause only 2ν individual transmissions. In our analysis, we do not consider such attacks to be serious attacks, both because data forwarding functionality is typically not a part of routing protocols, and because the attacker can only cause a constant number of transmissions as a result of its single transmission.

5.3. Attacks on DSR

As described in Section 2, DSR divides the routing problem into the two mechanisms of Route Discovery and Route Maintenance, both of which are subject to attack. We consider here only attacks on the basic operation of these mechanisms; attacks on the optimizations defined for these mechanisms in the DSR protocol [35, 36, 37] are beyond the scope of this paper.

An example of an attack against Route Discovery is for an attacker to forge ROUTE REPLY packets, causing nodes to misroute packets and to add incorrect routes to their Route Caches; likewise, an attacker can modify the node list in a ROUTE REQUEST or ROUTE REPLY that it forwards, or can modify the source route in a data packet that it forwards. An attacker can also initiate frequent Route Discoveries as a resource consumption attack to cause congestion in the network. Another resource consumption attack is for an attacker to deliberately select very long, suboptimal routes from its own Route Cache when originating packets, in an attempt to cause a large number of other nodes in the network to expend resources forwarding the packet. An attacker can decline to forward ROUTE REQUEST packets from other nodes, yet may originate its own data and ROUTE REQUEST packets, in effect unfairly using the services of the network while not cooperating to provide services to other nodes [50]. If two attackers can form a wormhole between themselves, they could forward ROUTE REQUEST and ROUTE REPLY packets over the wormhole but decline to forward data packets over it; a source node may then be able to discover a route over the virtual link of the wormhole but will be unable to successfully use it for routing data packets. In the worst case, this attack may also prevent the Route Discovery from reaching the target over any routes not using the wormhole.

An example of an attack against Route Maintenance is for an attacker to forge ROUTE ERROR packets, causing nodes incorrectly remove routes from their Route Caches and to use suboptimal routes even when more optimal routes had been previously discovered; in the worst case, this attack could prevent a node from being able to route any packets

to selected destinations. An attacker that is an intermediate node along some route could also discontinue forwarding data packets without sending a ROUTE ERROR indicating a problem with the next link; this attack could create a black hole, as discussed in Section 5.2.

6. Ariadne

6.1. Design Goals

A secure ad hoc network routing protocol minimally must be secure against an ActiveX attacker. An ActiveX attacker may be able to jam network packets at the physical layer if the physical layer does not protect against this. An ActiveX attacker may also attempt to forge control packets. One way to protect against such forged packets is to use a network-wide MAC key above the physical layer, authenticating each packet before processing it. Unfortunately, this approach is not robust against a single compromised node. Ideally, the probability that the routing protocol delivers messages, degrades gracefully when nodes fail or are captured. We aim for robustness against ActiveC and ActiveCX, and resilience against ActiveCCX attackers. In Section 7.2, we discuss the security provided by Ariadne in more detail.

Authentication of routing messages, allowing the receiver to identify the sender, helps to guard against an ActiveX attacker. Authentication also ensures *integrity*, so no adversary can alter the message in transit. We need an authentication mechanism that is compact, adding little overhead to the size of packets. The authentication mechanism should also be computationally inexpensive to generate and to verify, since otherwise an attacker can easily perform a Denial-of-Service (DoS) attack by flooding the nodes with malicious messages, overwhelming nodes that must attempt to verify the authentication. Thus, for point-to-point authentication of a message, we use a message authentication code (MAC) (e.g., HMAC [6]) and a shared key between the two parties. However, for authentication of a broadcast packet such as a ROUTE REQUEST, we cannot use a simple MAC, since any one receiver that knows the MAC key can alter the content and re-compute the MAC. We instead use the TESLA [65, 66] broadcast authentication protocol, which we review in Section 3.

In general, ad hoc network routing protocols do not need *secrecy* or *confidentiality*. These properties are required to achieve privacy or anonymity for the sender of messages. Even in the Internet, it is challenging to achieve sender anonymity, and this area is still the subject of active research [9, 14, 74, 75].

Our protocol does not protect regular data packets, so an Active1 attacker can always send malicious packets with long routes, causing nodes to expend resources forwarding them. If we limit the maximum route length to ν hops, an attacker can send a message that causes at most 2ν transmissions (ν transmissions to route the packet towards the destination, and if it encounters a broken link, ν transmissions to then route ROUTE ERROR back to the sender). We consider denial-of-service (DoS) attacks where the ratio between the total work performed by nodes in the network and the work performed by the attacker is on the order of the number of nodes in the network. We prevent DoS attacks where the attacker sends a single packet that results in a packet flood throughout the network. As a special case, we protect specifically against this in the propagation of a ROUTE REQUEST message, which is the only type of message for which we require a flood in the routing protocol.

6.2. Notation

We use the following notation to describe security protocols and cryptographic operations:

- A, B are principals, such as communicating nodes.
- K_{AB} and K_{BA} denote the secret MAC keys shared between A and B (one key for each direction of communication).
- $\text{MAC}_{K_{AB}}(M)$ denotes the computation of the message authentication code (MAC) of message M with the MAC key K_{AB} , for example HMAC [6].

6.3. Basic Ariadne Route Discovery

We assume that every end-to-end communicating source-destination pair of nodes A and B share the MAC keys K_{AB} and K_{BA} . We also assume that every node has a TESLA one-way key chain, and that all nodes know an authentic key of the TESLA one-way key chain of each other node (for authentication of subsequent keys, as described in Section 3). Route Discovery has two stages: the initiator floods the network with a ROUTE REQUEST, and the target returns the ROUTE REPLY. To secure the ROUTE REQUEST packet, our protocol provides the following properties: (1) the target node can authenticate the initiator (using a MAC with a key shared between the initiator and the target); (2) the initiator

can authenticate each entry of the path in the ROUTE REPLY (each intermediate node appends a MAC with its TESLA key); and (3) no intermediate node can remove a previous node in the node list in the REQUEST or REPLY (a one-way function prevents a compromised node from removing a node from the node list).

A ROUTE REQUEST packet in Ariadne contains eight fields: \langle ROUTE REQUEST, *initiator*, *target*, *id*, *time interval*, *hash chain*, *node list*, *MAC list* \rangle . The *initiator* and *target* are set to the address of the initiator and target nodes, respectively. As in DSR, the initiator sets the *id* to an identifier that it has not recently used in initiating a Route Discovery. The *time interval* is the TESLA time interval at the pessimistic expected arrival time of the REQUEST at the target, accounting for clock skew; specifically, given τ , a pessimistic transit time⁴, the time interval could be set to any time interval for which the key is not released within the next $\tau + 2\Delta$ time⁵. The initiator of the REQUEST then initializes the *hash chain* to $\text{MAC}_{K_{SD}}(\textit{initiator}, \textit{target}, \textit{id}, \textit{time interval})$ and the *node list* and *MAC list* to empty lists.

When any node *A* receives a ROUTE REQUEST for which it is not the target, the node checks its local table of \langle *initiator*, *id* \rangle values from recent REQUESTS it has received, to determine if it has already seen a REQUEST from this same Route Discovery. If it has, the node discards the packet, as in DSR. The node also checks whether the *time interval* in the REQUEST is valid: that time interval must not be too far in the future, and the key corresponding to it must not have been disclosed yet. If the time interval is not valid, the node discards the packet. Otherwise, the node modifies the REQUEST by appending its own address (*A*) to the *node list* in the REQUEST, replacing the *hash chain* field with $H[A, \textit{hash chain}]$, and appending a MAC of the entire REQUEST to the *MAC list*. The node uses the TESLA key K_{A_i} to compute the MAC, where *i* is the index for the time interval specified in the REQUEST. Finally, the node rebroadcasts the modified REQUEST, as in DSR.

When the target node receives the ROUTE REQUEST, it checks the validity of the REQUEST by determining that the keys from the time interval specified have not been disclosed yet, and that the *hash chain* field is equal to

$$H[\eta_n, H[\eta_{n-1}, H[\dots, H[\eta_1, \text{MAC}_{K_{SD}}(\textit{initiator}, \textit{target}, \textit{id}, \textit{time interval})] \dots]]]$$

where η_i is the node address at position *i* of the *node list* in the REQUEST, and where *n* is the number of nodes in the *node list*. If the target node determines that the REQUEST is valid, it returns a ROUTE REPLY to the initiator, containing eight fields: \langle ROUTE REPLY, *target*, *initiator*, *time interval*, *node list*, *MAC list*, *target MAC*, *key list* \rangle . The *target*, *initiator*, *time interval*, *node list*, and *MAC list* fields are set to the corresponding values from the ROUTE REQUEST, the *target MAC* is set to a MAC computed on the preceding fields in the REPLY with the key K_{DS} , and the *key list* is initialized to the empty list. The ROUTE REPLY is then returned to the initiator of the REQUEST along the source route obtained by reversing the sequence of hops in the *node list* of the REQUEST.

A node forwarding a ROUTE REPLY waits until it is able to disclose its key from the time interval specified, then it appends its key from that time interval to the *key list* field in the REPLY and forwards the packet according to the source route indicated in the packet. Waiting delays the return of the ROUTE REPLY but does not consume extra computational power.

When the initiator receives a ROUTE REPLY, it verifies that each key in the key list is valid, that the *target MAC* is valid, and that each MAC in the *MAC list* is valid. If all of these tests succeed, the node accepts the ROUTE REPLY; otherwise, it discards it. Figure 2 shows an example of Route Discovery in Ariadne.

6.4. Basic Ariadne Route Maintenance

As with Route Discovery, Route Maintenance in Ariadne is based on DSR as described in Section 2. A node forwarding a packet to the next hop along the source route returns a ROUTE ERROR to the original sender of the packet if it is unable to deliver the packet to the next hop after a limited number of retransmission attempts. Our goal in securing this basic Route Maintenance operation is to prevent the injection of invalid ROUTE ERRORS into the network from any node other than the one on the sending end of the broken link specified by the ERROR. For example, ERRORS for the link from node *B* to node *C* must only be accepted from node *B*. Our approach follows the following idea. The node that encounters the broken link adds TESLA authentication information to the ROUTE ERROR, such that all nodes on the return path can authenticate the ERROR. All nodes on the return path to the source forward the ERROR. However, TESLA authentication is delayed, so all the nodes on the return path buffer the ERROR, but do not consider the error until it is authenticated. Later, the node that encountered the broken link discloses the key and sends it over the return path, which enables nodes on the return path to authenticate the buffered ERROR.

⁴The choice of τ can be adaptive, since a failed Route Discovery could indicate that τ was chosen too small. In addition, the target of the Discovery could provide feedback in the ROUTE REPLY when τ was chosen too long.

⁵The value 2Δ is used here for the same reasons as in Section 3.

$$\begin{aligned}
S : & \quad h_0 = \text{MAC}_{K_{SD}}(\text{REQUEST}, S, D, id, ti) \\
S \rightarrow * : & \quad \langle \text{REQUEST}, S, D, id, ti, h_0, (), () \rangle \\
A : & \quad h_1 = H[A, h_0] \\
& \quad M_A = \text{MAC}_{K_{Ati}}(\text{REQUEST}, S, D, id, ti, h_1, (A), ()) \\
A \rightarrow * : & \quad \langle \text{REQUEST}, S, D, id, ti, \underline{h_1}, \underline{(A)}, \underline{(M_A)} \rangle \\
B : & \quad h_2 = H[B, h_1] \\
& \quad M_B = \text{MAC}_{K_{Bti}}(\text{REQUEST}, S, D, id, ti, h_2, (A, B), (M_A)) \\
B \rightarrow * : & \quad \langle \text{REQUEST}, S, D, id, ti, \underline{h_2}, \underline{(A, B)}, \underline{(M_A, M_B)} \rangle \\
C : & \quad h_3 = H[C, h_2] \\
& \quad M_C = \text{MAC}_{K_{Cti}}(\text{REQUEST}, S, D, id, ti, h_3, (A, B, C), (M_A, M_B)) \\
C \rightarrow * : & \quad \langle \text{REQUEST}, S, D, id, ti, \underline{h_3}, \underline{(A, B, C)}, \underline{(M_A, M_B, M_C)} \rangle \\
D : & \quad M_D = \text{MAC}_{K_{DS}}(\text{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C)) \\
D \rightarrow C : & \quad \langle \text{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), \underline{M_D}, () \rangle \\
C \rightarrow B : & \quad \langle \text{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, \underline{(K_{Cti})} \rangle \\
B \rightarrow A : & \quad \langle \text{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (K_{Cti}, \underline{K_{Bti}}) \rangle \\
A \rightarrow S : & \quad \langle \text{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (K_{Cti}, K_{Bti}, \underline{K_{Ati}}) \rangle
\end{aligned}$$

Figure 2: Route Discovery example in Ariadne. The initiator node S is attempting to discover a route to the target node D . The bold underlined font indicates changed message fields relative to the previous message of that type.

A ROUTE ERROR packet in Ariadne contains six fields: $\langle \text{ROUTE ERROR}, \textit{sending address}, \textit{receiving address}, \textit{time interval}, \textit{error MAC}, \textit{recent TESLA key} \rangle$. The *sending address* is set to the address of the intermediate node encountering the error, and the *receiving address* is set to the intended next hop destination of the packet it was attempting to forward. In the example above, where node B is attempting to forward a packet to the next hop node C , if B is unable to deliver the packet to C , node B sends a ROUTE ERROR to the original sender of the packet, with the *sending address* set to B and the *receiving address* set to C . The *time interval* in the ROUTE ERROR is set to the TESLA time interval at the pessimistic expected arrival time of the ERROR at the destination, and the *error MAC* field is set to the MAC of the preceding fields of the ROUTE ERROR, computed using the sender of the ROUTE ERROR's TESLA key for the time interval specified in the ERROR. The *recent TESLA key* field in the ROUTE ERROR is set to the most recent TESLA key that can be disclosed for the sender of the ROUTE ERROR. We use TESLA for authenticating ROUTE ERRORS so that forwarding nodes can also authenticate and process the ERROR.

When sending a ROUTE ERROR, the destination of the packet is set to the source address of the original packet triggering the ERROR, and the ERROR is forwarded toward this node in the same way as a normal data packet; the source route used in sending the ROUTE ERROR packet is obtained by reversing the source route from the header of the packet triggering the ERROR. Each node that is either the destination of the ERROR or forwards the ERROR searches its Route Cache for all routes it has stored that use the $\langle \textit{sending address}, \textit{receiving address} \rangle$ link indicated by the ERROR. If the node has no such routes in its Cache, it does not process the ROUTE ERROR further (other than forwarding the packet, if it is not the destination of the ERROR). Otherwise, the node checks whether the *time interval* in the ERROR is valid: that time interval must not be too far into the future, and the key corresponding to it must not have been disclosed yet; if the time interval is not valid, the node similarly does not process the ROUTE ERROR further.

If all of the tests above for the ROUTE ERROR succeed, the node checks the authentication on the ERROR, based on the sending node's TESLA key for the time interval indicated in the ERROR. To do so, the node saves the information from the ROUTE ERROR in memory until it receives a disclosed TESLA key from the sender that allows this. During this time, the node continues to use the routes in its Route Cache without modification from this ERROR. If the error is transient or if the sender stops using that route, there will be no need to complete the authentication of the ROUTE ERROR. Otherwise, each subsequent packet sent along this route by this node will trigger an additional ROUTE ERROR, and once the TESLA time interval used in the first ERROR ends, the *recent TESLA key* field in the next ERROR returned will allow authentication of this first ERROR; alternatively, the node could also explicitly request the needed TESLA key from the sender once the interval ends. Once the ROUTE ERROR has been authenticated, the

node removes from its Route Cache all routes using the indicated link, and also discards any saved information for other ROUTE ERRORS for which, as a result of removing these routes, it then has no corresponding routes in its Route Cache.

To handle the possible memory consumption attack of needing to save information from many pending ROUTE ERRORS, the technique proposed by Kuhn [44] is quite effective. Specifically, a node receiving a ROUTE ERROR must wait for the disclosure of the TESLA key used by the sender to sign it; each node keeps in memory a table containing the information from each such received ROUTE ERROR that is waiting for authentication. We manage this table such that the probability that the information from a ROUTE ERROR is in the table is independent of the time that this node received that ERROR.

When the wireless link capacity is finite, an attacker can inject only a finite number of ROUTE ERRORS within a TESLA time interval plus $2\Delta + \tau$. As a result, the probability of success for our defense against memory consumption attacks for received ROUTE ERRORS in any time interval is given by $p_s = 1 - (y/(x + y))^N$, where N is the number of ROUTE ERRORS that can be held in the node's table, x is the number of authentic ROUTE ERRORS received, and y is the number of ERRORS sent by the attacker. The maintenance of a link therefore follows a geometric distribution, and the expected number of time intervals before success is $(1 - (y/(x + y))^N)^{-1}$. For example, in a network using a 1-second TESLA time interval and an 11 Mbps wireless link, if the size of a ROUTE ERROR packet is 60 bytes, then a node with a 5000-element table receiving just one authentic ROUTE ERROR per second can successfully authenticate and process one of the authentic ROUTE ERRORS within 5.1 seconds on the average, even when an attacker is otherwise flooding the node with malicious ROUTE ERRORS. This 5.1 second recovery time represents a *worst-case* scenario, and minimal node resources are consumed while the node waits to validate one of these ROUTE REQUESTS.

6.5. Thwarting Effects of Routing Misbehavior

The protocol described above is vulnerable to an ActiveC attacker that happens to be along the discovered route. In particular, we have not presented a means of determining whether intermediate nodes are in fact forwarding packets that they have been requested to forward. Watchdog and pathrater [50] attempt to solve this problem by identifying the attacking nodes and avoiding them in the routes used. Instead, we choose routes based on their prior performance in packet delivery. Introducing mechanisms that penalize specific nodes for routing misbehavior (such as is done in watchdog and pathrater) is subject to a blackmail attack (Section 5.1), where a sufficient number of attackers may be able to penalize a well-behaved node.

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 SACK [51]; this feedback approach is somewhat similar that used in IPv6 for Neighbor Unreachability Detection [58]. Stronger properties are obtained when the routing protocol sends such feedback packets along a route equal to the reversed route of the triggering packet; otherwise, a malicious node along one route may drop the acknowledgment for a packet transmitted along a functioning route.

A node with multiple routes to a single destination can assign a fraction of packets that it originates to be sent along each route. When a substantially smaller fraction of packets sent along any particular route are successfully delivered, the node can begin sending a smaller fraction of its overall packets to that destination along that route. However, if the fraction of packets chosen to be sent along a route that appears to be misbehaving were to reach zero, a short-lived jamming attack that is now over could still prevent the future use of that route. To avoid this possible Denial-of-Service attack, we choose the fraction of packets sent along such a route to be some small but nonzero amount, to allow the occasional monitoring of the route. We use normal data packets for this infrequent monitoring rather than sending separate "probe" packets, in order to avoid an attack in which some node might forward only probes and not data; this approach could also be extended to use probe packets disguised to look to an attacker like normal data packets, to avoid possibly losing even infrequent data packets on suspect routes.

Because DSR often returns multiple ROUTE REPLY packets in response to a Route Discovery, the presence of multiple routes to some destination in a node's Route Cache is quite common. Tsirigos and Haas [86] also discuss the use of multiple routes for increasing reliability, although they do not discuss this technique with respect to secure routing protocols.

6.6. Thwarting Malicious Route Request Floods

An active attacker can attempt to degrade the performance of DSR or other on-demand routing protocols by repeatedly initiating Route Discovery. In this attack, an attacker sends ROUTE REQUEST packets, which the routing protocol floods throughout the network. In basic Ariadne, as described in Sections 6.3 and 6.4, a ROUTE REQUEST is not

authenticated until it reaches its target, thus allowing an **Active1** attacker to cause such network-wide floods. Even if all packets were authenticated with a network-wide authentication key, an **ActiveC** attacker could still flood the network.

To protect Ariadne from a flood of **ROUTE REQUEST** packets, we need a mechanism that enables nodes to instantly authenticate **ROUTE REQUESTS**, so nodes can filter out forged or excessive **ROUTE REQUEST** packets. We introduce *Route Discovery chains*, a mechanism for authenticating Route Discoveries, allowing each node to rate-limit Discoveries initiated by any node.

Route Discovery chains are one-way chains generated, as in **TESLA** (Section 3), by choosing a random K_N , and repeatedly computing a one-way hash function H to give $K_i = H^{N-i}[K_N]$. These chains can be used in one of two ways. One approach is to release one key for each Route Discovery. Each **ROUTE REQUEST** from that Discovery would carry a key from this Route Discovery chain, and duplicates could be suppressed using this value. Because of the flooding nature of Route Discovery, a node that is not partitioned from the network will generally hear each chain element that is used, preventing an attacker from reusing that value in the future. An alternative approach, similar to **TESLA**, is to dictate a schedule at which Route Discovery chain elements can be used, and to use loosely synchronized clocks to prevent even partitioned nodes from propagating an old **ROUTE REQUEST**. The latter approach is slightly more computationally expensive, but is secure against an attacker replaying an old chain element to a formerly partitioned node, causing that node to ignore **REQUESTS** from the spoofed source for some period of time.

6.7. Establishing Authenticated Keys using a KDC

Although Ariadne does not require a trusted Key Distribution Center (KDC) node in the ad hoc network, some ad hoc networks may contain a KDC as a mechanism for authenticated key setup between pairs of nodes as need, as mentioned in Section 4.3. A challenge to this type of key setup is the need for routing to be established before conventional KDC protocols can be used; we describe here how to do this type of key setup with no established routing within Ariadne, if desired.

We assume that every node A shares MAC keys with a trusted KDC, as described in Section 6.2, and can obtain an authentic **TESLA** key of the KDC. The KDC must also have an authentic **TESLA** key for each other node. We also assume that each node shares two encryption keys with the KDC. Finally, if Route Discovery chains are used to thwart **ROUTE REQUEST** floods as described in Section 6.6, each node must have an authentic element of the KDC's Route Discovery chain.

To bootstrap authenticated keys between pairs of nodes, the KDC node initiates a Route Discovery with a special, reserved address (not the address of any actual node) as the target of the Discovery. The Route Discovery is processed as in Ariadne, except that each node receiving the **REQUEST** for the first time also returns a **ROUTE REPLY**. The KDC can then use each returned route to send encrypted, authenticated keys to each node in the network. Alternatively, the Route Discovery process can be repeated periodically. When a node needs a shared key with some other node, it waits until the next such Discovery is initiated by the KDC node, and at that time includes as part of its **ROUTE REPLY** the list of nodes for which it needs authenticated keys. The KDC can then generate the needed keys, and send them, encrypted, to the two communicating parties.

7. Performance Evaluation

7.1. Simulation Evaluation

To evaluate the basic version of Ariadne without attackers, we used the *ns-2* simulator, with mobility extensions from the Monarch project [12]. The *ns-2* simulator has been used extensively in evaluating performance of ad hoc network routing protocols. Our simulations model radio propagation using the realistic *two-ray ground reflection* model [73] and account for physical phenomena such as signal strength, propagation delay, capture effect, and interference. The Medium Access Control protocol used is the IEEE 802.11 Distributed Coordination Function (DCF) [33]. The parameters used for the simulation are given in Table 1.

We modeled Ariadne by modifying DSR in several ways: we increased the packet sizes to reflect the additional fields necessary for authenticating the packets and modified the handling of Route Discovery and Maintenance for the additional authentication processing defined in Ariadne; we adjusted retransmission timeouts for **ROUTE REQUESTS** to compensate for the delay necessary for the disclosure of **TESLA** keys; and we treated routes learned from Route Discovery in an atomic fashion that did not allow the use of prefixes of routes in the Route Cache. We compare

Table 1: Parameters for Ariadne Simulations

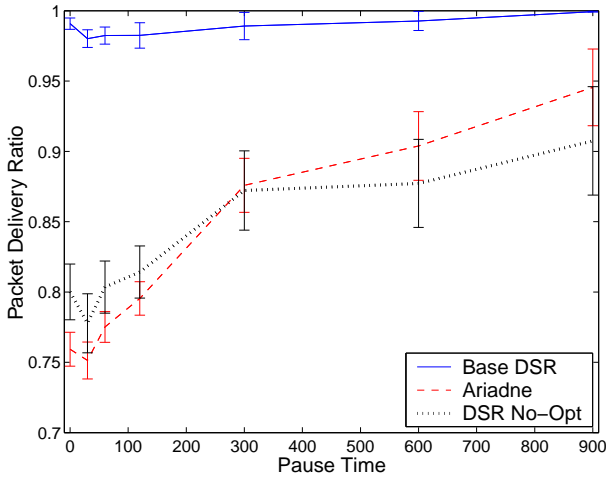
Scenario Parameters	
Number of Nodes	50
Maximum Velocity (v_{\max})	20 m/s
Dimensions of Space	1500 m \times 300 m
Nominal Radio Range	250 m
Source-Destination Pairs	20
Source Data Pattern (each)	4 packets/second
Application Data Payload Size	512 bytes/packet
Total Application Data Load	327 kbps
Raw Physical Link Bandwidth	2 Mbps
DSR Parameters	
Initial ROUTE REQUEST Timeout	2 seconds
Maximum ROUTE REQUEST Timeout	40 seconds
Cache Size	32 routes
Cache Replacement Policy	FIFO
TESLA Parameters	
TESLA Time Interval	1 second
Pessimistic End-to-End Propagation Time (τ)	0.2 seconds
Maximum Time Synchronization Error (Δ)	0.1 seconds
Hash Length (ρ)	80 bits

Ariadne here with the current version of DSR [29], which we call “Base DSR,” and with an unoptimized version of DSR, which we call “DSR No-Opt.” In DSR No-Opt, we disabled all protocol optimizations not present in Ariadne. By comparing Ariadne with this unoptimized version of DSR, we can examine the performance impact of adding security, independent of the performance impact of the DSR optimizations removed to allow the security changes.

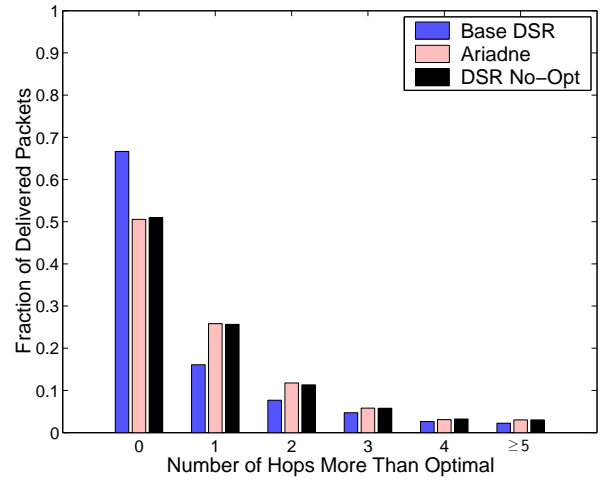
Each node in our simulation moves according to the *random waypoint* model [36]: a node starts at a random position, waits for a duration called the *pause time*, and then chooses a new random location and moves there with a velocity uniformly chosen between 0 and v_{\max} . When it arrives, it waits for the pause time and repeats the process. Like much previous work in evaluating ad hoc network routing protocols (e.g., [12, 29, 34]), we use a rectangular space of size 1500 m \times 300 m to increase the average number of hops in routes used relative to a square space of equal area, creating a more challenging environment for the routing protocol. All protocols were run on identical movement and communication scenarios. We computed five metrics for each simulation run:

- *Packet Delivery Ratio*: The total fraction of application-level data packets sent that are actually received at the intended destination node.
- *Packet Overhead*: The number of transmissions of routing packets; for example, a ROUTE REPLY sent over three hops would count as three packets in this metric.
- *Byte Overhead*: The number of transmissions of overhead (non-data) bytes.
- *Mean Latency*: The average time elapsed from when a data packet is first sent to when it is first received at its destination.
- *99.99th Percentile Latency*: Computed as the 99.99th percentile of the packet delivery latency
- *Path Optimality*: Compares the length of routes used to the optimal hop length as determined by an off-line omniscient algorithm.

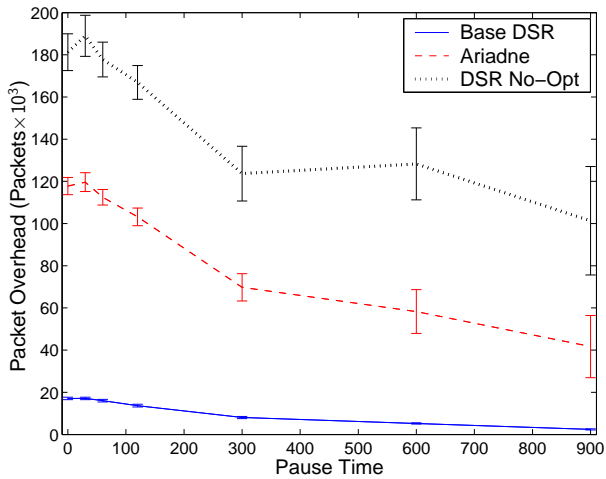
Figure 3(a) shows the Packet Delivery Ratio (PDR) for each protocol. Removing the optimizations from Base DSR to produce DSR No-Opt reduces PDR by between an average of 15.2%; adding Ariadne security further reduces PDR by just an additional 0.66% on average, and does not reduce PDR by more than an additional 4% at any pause time. Ariadne delivers fewer packets than DSR No-Opt at higher levels of mobility for two reasons: first, since Route Discovery operates more slowly, packets are more likely to time out waiting for a ROUTE REPLY, and the route contained in a ROUTE REPLY will have a shorter lifetime; and second, because ROUTE ERRORS cannot be processed until the TESLA key used is disclosed, additional data packets continue to be sent along the broken route for on average half of the TESLA time interval after the ERROR is received.



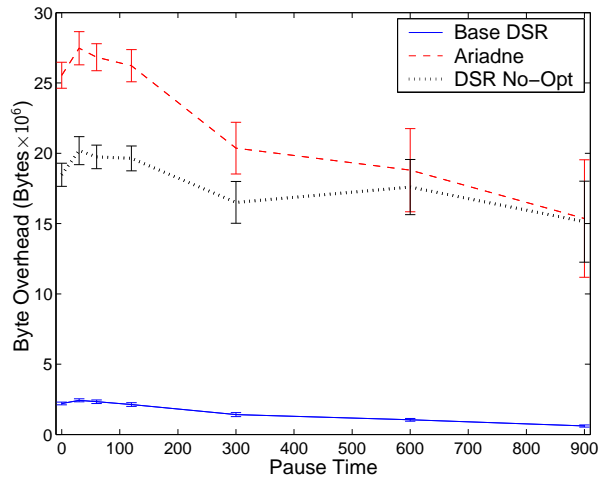
(a) Packet Delivery Ratio



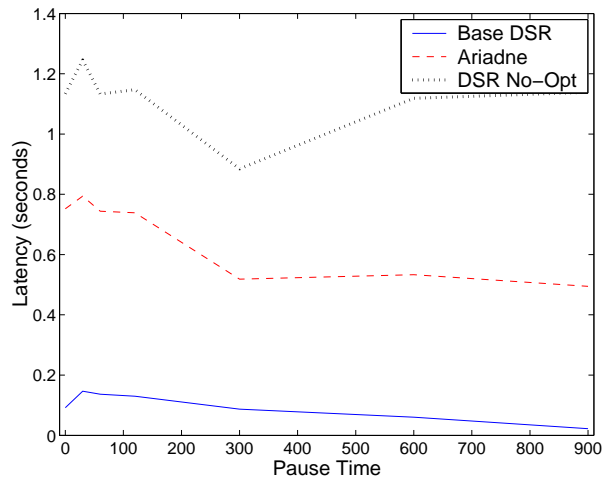
(b) Path Optimality



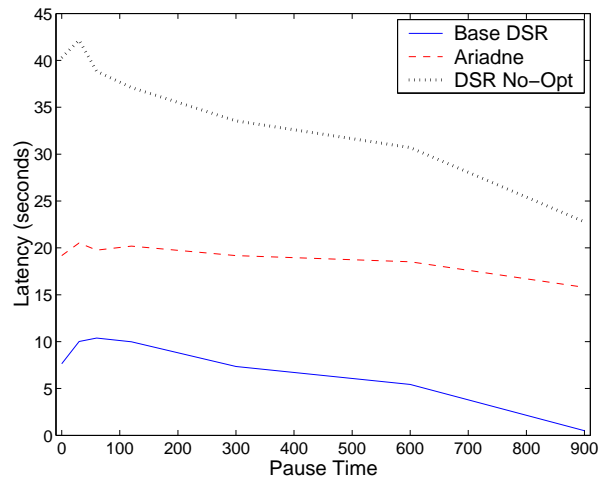
(c) Packet Overhead



(d) Byte Overhead



(e) Average Latency



(f) 99.99th Percentile Latency

Figure 3: Ariadne performance evaluation results, based on simulation over 60 runs; the error bars represent the 95% confidence interval of the mean.

Surprisingly, Ariadne actually outperforms DSR No-Opt at lower levels of mobility. This improved performance results from the average half-second delay (one half of the TESLA time interval) that Ariadne introduces between the target receiving a ROUTE REQUEST and sending a ROUTE REPLY. Specifically, when a REQUEST traverses a short-lived link, DSR No-Opt immediately returns the REPLY, but the new route can be used for only its brief lifetime, contributing additional overhead for forwarding the REPLY and for sending and forwarding the ERROR. In Ariadne, links are tested twice: once when the REQUEST traverses the network, and once when the REPLY is sent along the reverse path. If one of these links breaks between these tests, the REPLY with this route is not received by the initiator. It is this additional route confirmation that allows Ariadne to find more stable routes than DSR No-Opt.

Figure 3(b) shows Path Optimality. In Base DSR, the average number of hops along a route used by a packet is 0.6853 more than the minimum possible, based on the nominal wireless transmission range of 250 m per hop. In DSR No-Opt, routes used are on average 0.2705 hops longer than in Base DSR, and in Ariadne, routes used average 0.0044 hops longer than in DSR No-Opt. DSR No-Opt performs slightly better than Ariadne because it initiates more Route Discoveries, and thus tends to find shorter routes more quickly than in Ariadne.

Figures 3(c) and 3(d) show the packet and byte overhead, respectively. Ariadne has *consistently lower* packet overhead than DSR No-Opt, because Ariadne tends to find more stable routes than DSR No-Opt, reducing the number of ROUTE ERRORS that are sent. This advantage is somewhat countered by the increase in number of ROUTE ERRORS used by Ariadne: since ERROR processing is delayed, more redundant ERRORS are sent. Unfortunately, byte overhead in Ariadne is significantly worse than in either Base DSR or DSR No-Opt, due to the authentication overhead in ROUTE REQUEST, REPLY, and ERROR packets.

Figures 3(e) and 3(f) show the average and 99.99th percentile latency for the protocols. Because of the reduced number of broken links that get used in Ariadne relative to DSR No-Opt, Ariadne generally has better latency than DSR No-Opt.

7.2. Security Analysis

In this section, we discuss how Ariadne resists attacks by certain attacker types, according to the taxonomy we present in Section 5.1.

To characterize the security offered by Ariadne, we define a minimum broadcast latency path between a source and a destination to be any path that forwards a Route Discovery most quickly from the source to the destination. We call a route that only consists of uncompromised nodes an *uncompromised route*. Ariadne prevents compromised nodes from disturbing uncompromised routes. In particular, Ariadne provides two properties assuming reliable broadcast:

- If there exists an uncompromised neighbor of a destination such that all minimum latency paths between the initiator of the Discovery and that neighbor are uncompromised routes, then an uncompromised route from the initiator to the target will be returned in a ROUTE REPLY.
- If at least one REPLY returned as a result of the first property represents a shortest route from the initiator to the target, Ariadne will use one such uncompromised route.

To argue for the correctness of the first property, we note that if all minimum latency paths between the initiator and a neighbor of the destination are uncompromised, then the first REQUEST to reach that neighbor comes over an uncompromised route. Since it is the first REQUEST, it will not be filtered by duplicate REQUEST detection, so it will be rebroadcast, and heard by the target. Since the target returns a REPLY for each REQUEST it receives, without performing duplicate detection, a REPLY will be returned. The second property trivially follows from the use of shortest paths and the first property.

Though reliable broadcast may not be securely or efficiently achievable, we assume that most broadcast packets are received, and hence the properties listed above generally hold.

We now consider Ariadne using the taxonomy of attacks that we presented in Section 5.1. We first address some generic attacks, then we list different attacker configurations in increasing strength, and discuss how Ariadne resists these attacks.

An attacker may attempt to creating a gray hole or black hole by removing nodes in a ROUTE REQUEST; however, the hash chain in each REQUEST prevents such tampering. An attacker may fabricate nodes to insert in the accumulated route list of a REQUEST packet, such fabricated nodes would not have known keys at the source, and the REPLY would thus not be authenticated. If the attacker tries to replace the MAC and keys in the reply, such tampering will be detected as a result of the *target MAC* field in the REPLY. Because Ariadne uses source routes, and because it does not contain a mechanism for legitimate nodes to change the source route once a packet is in transit, an attacker cannot cause non-attacking nodes to form a routing loop. An attacker can create a routing loop by including itself in the source route; this behavior, however, is no worse than if the attacker were to source packets with period equal to the propagation

time around the loop. In particular, if there are n nodes in the routing loop, and a single packet is forwarded around the loop m times, the attacker participates in m forwards, and the total expended effort is mn forwards. Had the attacker instead sourced m packets along n -hop routes, the total attacker effort is m transmissions, and the total network effort is mn forwards, an identical result.

Since Ariadne does not attempt to provide anonymous routing, a **Passive** attacker can eavesdrop on all routing traffic and learn which parties communicate.

One attacking node (an **Active1** attacker) may attempt the following attacks:

- Send bogus **ROUTE REQUESTS**. Since the attacker does not have a registered one-way Route Discovery chain, its neighbors will immediately detect and reject the bogus packets. If the attacker eavesdrops on another **ROUTE REQUEST** and replays it (or an altered **REQUEST** with the same route request chain value), a neighbor will drop the **REQUEST** if it has already seen it; otherwise either the target of the Discovery will ignore the **REQUEST** flowing through this attacker, or the initiator will drop the **REPLY** corresponding to that **REQUEST**.
- Replay **ROUTE REQUEST**, **ROUTE REPLY**, and **ROUTE ERROR** messages. Ariadne ensures freshness by using authenticated identifiers, and rejecting messages that do not have a fresh identifier. However, this does not prevent an attacker from instantly replaying that message, effectively making it behave as a repeater. In prior work, we designed **TIK** [67], which can even prevent such an attack, but it relies on highly accurate clock synchronization.
- Add or remove nodes from a **ROUTE REQUEST**. If an attacker removes one or more nodes from the accumulated route list, the target can verify that the hash chain is inconsistent with the bogus route list. If an attacker adds bogus nodes to the accumulated route list, the initiator can detect the attack, because the attacker cannot authenticate the response with **TESLA**. If the attacker changes the nodes to legitimate ones when it receives the **REPLY**, the target **MAC** will reveal tampering.
- Change basic packet information in a **ROUTE REQUEST**. The integrity of the basic packet information, such as the initiator and target address, identifier, and time interval, is protected with a **MAC** generated by the initiator, which the target checks.

Multiple attackers (an **ActiveX** attacker) may act as a wormhole, as described in Section 5.2. The wormhole attack may partially disrupt routing. The basic Ariadne protocol does not prevent this attack, but **TIK** [67] can secure against it.

An **ActiveC** attacker, with one compromised node, can attempt to flood network with many **ROUTE REQUESTS**. Since the source address of each **ROUTE REQUEST** is authenticated, and since each new Route Discovery needs to carry a new one-way Route Discovery chain value, the compromised node can only produce **ROUTE REQUESTS** with its own source address. Simple rate limiting of **ROUTE REQUESTS** at each node enforce an upper bound on the sending rate.

An **ActiveCX** attacker, with multiple attacking nodes equipped with the keys of single compromised node may attempt to construct a wormhole, but append the address and key of the compromised node in each **REQUEST** forwarded across this wormhole. **TIK** itself cannot prevent this attack, but **TIK** and **GPS** can be used in conjunction to ensure that an **ActiveCX** wormhole attack can be no worse than an **ActiveC** attacker positioned correctly. In particular, if each node forwarding a **ROUTE REQUEST** includes its alleged **GPS** coordinates in that **REQUEST**, then a node can detect if it should be reachable from the previous hop, and if the hop before the previous hop should be able to reach the previous hop. If both of these checks succeed, then the attacker could have placed the compromised node at the position it specified in the packet, and that node would have been able to hear the original **REQUEST**, append its address, and forward it to the next hop.

Multiple attackers that know all the keys of multiple nodes (an **ActiveCCX** attacker configuration) may perform the following attacks:

- Lengthen the route in the **REQUEST** by adding other compromised nodes to the route. If the source finds a shorter route, it will likely prefer that route, so the protocol behaves as if the attacker were not there.
- Attempt to force the initiator to repeatedly initiate Route Discoveries. Suppose an **ActiveCCX** attacker had the keys of multiple compromised nodes, and that one such attacker were on the shortest path from the source to the destination. When the attacker receives its first **ROUTE REQUEST** packet as part of some Discovery, it adds its address and **MAC**, as normal, but also adds the address of another node it has compromised. When data packets are sent along that route, the attacker replies with a **ROUTE ERROR** from its first hop to its second hop. In subsequent Route Discoveries, the attacker can use different addresses for the additional address. Since other routes may have been returned as part of any of these Route Discoveries, this attack is not guaranteed to be successful.

To prevent such starvation, the initiator may include data in the ROUTE REQUEST. To be part of the path, the attacker must forward routing messages, so the initiator can send data to the target. If the attacker alters the data in the ROUTE REQUEST, the destination will detect the alteration (using the shared key and a MAC on the data) and reject that route.

A set of attackers that control a vertex cut of the network, or an ActiveVC attacker, may perform the following additional attacks:

- Make nodes on one side of the vertex cut believe that any node on the other side is attempting to flood the network. By holding and not propagating ROUTE REQUESTS from a certain node for some time, then initiating many Route Discoveries with the chain values from the old Discoveries, an ActiveVC attacker can make that node appear to be flooding the network. When the use of individual elements of a Route Discovery chain are time-synchronized, this attack simply causes the REQUESTS associated with the stale chain elements to be discarded.
- Only forward ROUTE REQUEST and ROUTE REPLY packets. A sender is then unable to successfully deliver packets. This attack is only marginally different from not participating in the protocol at all, differing only in that the sender and some intermediate nodes continue to spend power to send packets, but none of those packets are successfully received.

8. Related Work

Several researchers have proposed secure routing protocols. For example, Perlman [64] proposed *flooding NPBR*, an on-demand protocol designed for wired networks that floods each packet through the network. To prevent denial of service attacks, flooding NPBR allocates a fraction of the bandwidth along each link to each node, and verifies the authenticity of packets by using digital signatures. Unfortunately, this protocol has high overhead in terms of both computational resources necessary for digital signature verification and in terms of bandwidth requirements. Furthermore, estimating and guaranteeing available bandwidth in a wireless environment is difficult [41].

Zhou and Haas [90], Zapata [88], and Dahill et al. [19] propose the use of asymmetric cryptography to secure on-demand ad hoc network routing protocols. However, when the nodes in an ad hoc network are generally unable to verify asymmetric signatures quickly enough, or when network bandwidth is insufficient, these protocols may not be suitable. A number of research groups have developed protocols for securing periodic protocols based on asymmetric cryptography, such as Kent et al. [40], Perlman's *link-state NPBR*, Kumar's secure link-state protocol [45], and Smith et al. [82, 83]. In addition to the disadvantages of on-demand protocols based on asymmetric cryptography, these protocols may suffer in some scenarios because periodic protocols may not be able to cope with high rates of mobility. Kumar also discusses threats to both distance-vector protocols and link-state protocols, and describes techniques for securing distance-vector protocols. However, these techniques are vulnerable to the compromise of a single node.

Cheung [15] and Hauser et al. [24] describe symmetric-key approaches to the authentication of link-state updates, but do not discuss mechanisms for detecting the status of these links. In wired networks, a common technique for authenticating HELLO packets is to verify that the incoming interface is the expected interface and that the IP TTL of the packet is 255. In a wireless network, this technique cannot be used. Furthermore, these protocols assume the use of periodic routing protocols, which are not always suitable in ad hoc networks. Finally, Cheung [15] uses similar cryptographic mechanisms, but optimistically integrates routing data before it is authenticated, adversely affecting security.

A number of researchers have proposed the use of symmetric schemes for authenticating routing control packets. Heffernan [27] proposes a mechanism requiring shared keys between all communicating routers. This scheme may not scale to large ad hoc networks, and may be vulnerable to single-node compromise. Perrig et al. [68] use symmetric primitives to secure routing for routes between nodes and a trusted base station. Basagni et al. [5] use a network-wide symmetric key to secure routing communication, which is vulnerable to a single node compromise, although they specify the use of secure hardware to limit the damage that can be done by a compromised node. Papadimitratos and Haas [59] present work that secures against non-colluding adversaries, and do not authenticate intermediate nodes which forward ROUTE REQUESTS. Hu et al. [31] use hash chains to authenticate routing updates sent by a periodic protocol; however, their approach builds on a periodic protocol, and such approaches tend to have higher overhead than on-demand protocols.

Yi, Naldurg, and Kravets [87] present the notion of security requirements for paths carrying certain traffic, and briefly discuss mechanisms for securing routing, such as network-wide encryption of ROUTE REQUESTS and use of digital signatures.

Some researchers have explored the establishment of trust in ad hoc networks. Hubaux et al. [32] bootstrap trust relationships based on PGP-like certificates, assuming an existing routing infrastructure. Stajano and Anderson [84] propose a scheme for establishing trust and keys between two nodes in an ad hoc network. Balfanz et al. [4] generalize their approach, assuming a side channel in which detection of multiple transmitters is possible. Our techniques for bootstrapping, presented in Section 6.7, present an approach that requires minimal offline bootstrapping and no public key operations after initialization.

Routing protocol intrusion detection has been studied in wired networks as a mechanism for detecting misbehaving routers. Cheung and Levitt [16] and Bradley et al. [11] propose intrusion detection techniques for detecting and identifying routers that send bogus routing update messages. In this paper, we attempt to authenticate packets before processing them, instead of relying on the delayed reaction of an intrusion detection system.

Marti et al. [50] consider the problem of detecting intermediate nodes that do not forward packets. Our approach differs in that we choose routes that we have found to work, rather than relying on watching as nodes forward traffic from arbitrary sources.

9. Conclusions

This paper has presented the design and evaluation of Ariadne, a new ad hoc network routing protocol that provides security against one compromised node and arbitrary active attackers, and relies only on symmetric cryptography. Ariadne operates on demand, dynamically discovering routes between nodes only as needed; the design is based on the basic operation of the DSR protocol. Rather than generously applying cryptography to an existing protocol to achieve security, we carefully re-designed each protocol message and its processing.

Because we did not secure the optimizations of DSR in Ariadne, the resulting protocol is less efficient than the highly optimized version of DSR that runs in a trusted environment. However, we also compared Ariadne to a version of DSR in which we disabled all protocol optimizations not present in Ariadne, allowing us to evaluate and analyze the effect of the optimizations and the security separately. The byte overhead of Ariadne was 26.19% higher than for this unoptimized version of DSR due to the overhead of the authentication information in Ariadne's routing packets. As explained in our results, however, Ariadne actually performs *better* on some metrics (e.g., 41.7% lower packet overhead) than this version of DSR, and about the same on all other metrics, even though Ariadne must bear the added costs for security not present in DSR.

We found that the source-routing in DSR greatly facilitated adding security. Source routing empowers the sender to circumvent potentially malicious nodes, and enables the sender to authenticate of every node in a ROUTE REPLY. Such fine-grained path control is absent in most distance-vector routing protocols, which makes security much more difficult to achieve.

References

- [1] Norman Abramson. The ALOHA system—another alternative for computer communications. In *Proceedings of the Fall 1970 AFIPS Computer Conference*, pages 281–285, November 1970.
- [2] Ross Anderson, Charalampos Maniavas, and Chris Sutherland. NetCard – a practical electronic cash system. In Lomas [47].
- [3] N. Asokan, Gene Tsudik, and Michael Waidner. Server-supported signatures. *Journal of Computer Security*, 5(1):91–108, 1997.
- [4] Dirk Balfanz, D.K. Smetters, Paul Stewart, and H. Chi Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Symposium on Network and Distributed Systems Security (NDSS '02)*, San Diego, California, February 2002.
- [5] Stefano Basagni, Kris Herrin, Emilia Rosti, and Danilo Bruschi. Secure Pebblenets. In *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001)*, pages 156–163, Long Beach, California, USA, October 2001.
- [6] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology - Crypto '96*, pages 1–15, Berlin, 1996. Springer-Verlag. Lecture Notes in Computer Science Volume 1109.
- [7] Bhargav Bellur and Richard G. Ogier. A reliable, efficient topology broadcast protocol for dynamic networks. In *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99)*, pages 178–186, March 1999.

- [8] A. Benjaminson and S. C. Stallings. A microcomputer-compensated crystal oscillator using a dual-mode resonator. In *Proceedings of the 43rd Annual Symposium on Frequency Control*, pages 20–26, May 1989.
- [9] Oliver Berthold, Hannes Federrath, and Marit Köhntopp. Project “Anonymity and Unobservability in the Internet”. In *Workshop on Freedom and Privacy by Design / CFP2000*, 2000.
- [10] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang. MACAW: A media access protocol for wireless LANs. In *Proceedings of the SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 212–225. ACM, August 1994.
- [11] Kirk A. Bradley, Steven Cheung, Nick Puketza, Biswanath Mukherjee, and Ronald A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 115–124, Oakland, CA, May 1998.
- [12] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta G. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 85–97. ACM/IEEE, October 1998.
- [13] Michael Brown, Donny Cheung, Darrel Hankerson, Julio Lopez Hernandez, Michael Kirkup, and Alfred Menezes. PGP in constrained wireless devices. In *9th USENIX Security Symposium*, August 2000.
- [14] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.
- [15] Steven Cheung. An efficient message authentication scheme for link state routing. In *13th Annual Computer Security Applications Conference*, 1997.
- [16] Steven Cheung and Karl Levitt. Protecting routing infrastructures from denial of service using cooperative intrusion detection. In *The 1997 New Security Paradigms Workshop*, September 1998.
- [17] Tom Clark. Tom Clark’s totally accurate clock ftp site. Greenbelt, Maryland. Available at <ftp://aleph.gsfc.nasa.gov/GPS/totally.accurate.clock/>.
- [18] Nicolas Courtois, Louis Goubin, and Jacques Patarin. Flash, a fast multivariate signature algorithm. In David Naccache, editor, *Progress in Cryptology — CT-RSA 2001*, number 2020 in LNCS. Springer-Verlag, Berlin Germany, April 2001.
- [19] Bridget Dahill, Brian Neil Levine, Elizabeth Royer, and Clay Shields. A secure routing protocol for ad hoc networks. Technical Report UM-CS-2001-037, Electrical Engineering and Computer Science, University of Michigan, August 2001.
- [20] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. In Gilles Brassard, editor, *Advances in Cryptology - Crypto '89*, pages 263–277, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 435.
- [21] Eran Gabber and Avishai Wool. How to prove where you are. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 142–149, November 1998.
- [22] Zygmunt J. Haas and Marc R. Pearlman. The performance of query control schemes for the zone routing protocol. In *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 167–177, June 1998.
- [23] Neil M. Haller. The S/KEY one-time password system. In *ISOC*, 1994.
- [24] Ralf Hauser, Antoni Przygienda, and Gene Tsudik. Reducing the cost of security in link state routing. In *Symposium on Network and Distributed Systems Security (NDSS '97)*, pages 93–99, San Diego, California, February 1997.
- [25] Ralf Hauser, Michael Steiner, and Michael Waidner. Micro-payments based on iKP. Research Report 2791 (# 89269), IBM Research, February 1996.
- [26] Ralf Hauser and Gene Tsudik. On shopping *incognito*. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, Oakland, California, November 1996. USENIX.
- [27] Andy Heffernan. Protection of BGP sessions via the TCP MD5 signature option. RFC 2385, August 1998.
- [28] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NSS: An NTRU lattice-based signature scheme. In Birgit Pfizmann, editor, *Advances in Cryptology — EUROCRYPT '2001*, number 2045 in Lecture Notes in Computer Science, pages 211–228. Springer-Verlag, Berlin Germany, 2001.
- [29] Yih-Chun Hu and David B. Johnson. Caching strategies in on-demand routing protocols for wireless ad hoc networks. In *Proceedings of the Sixth Annual IEEE/ACM International Conference on Mobile Computing and Networking (MobiCom 2000)*, pages 231–242. ACM, August 2000.

- [30] Yih-Chun Hu and David B. Johnson. Implicit source routing in on-demand ad hoc network routing. pages 1–10, October 2001.
- [31] Yih-Chun Hu, David B. Johnson, and Adrian Perrig. Secure efficient distance vector routing in mobile wireless ad hoc networks. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '02)*, June 2002. To appear.
- [32] Jean-Pierre Hubaux, Levente Buttyán, and Srdjan Čapkun. The quest for security in mobile ad hoc networks. In *ACM Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, CA, USA, October 2001.
- [33] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.
- [34] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based performance analysis of routing protocols for mobile ad-hoc networks. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, pages 195–206. ACM/IEEE, August 1999.
- [35] David B. Johnson. Routing in ad hoc networks of mobile hosts. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'94)*, pages 158–163, December 1994.
- [36] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Tomasz Imielinski and Hank Korth, editors, *Mobile Computing*, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.
- [37] David B. Johnson, David A. Maltz, Yih-Chun Hu, and Jorjeta G. Jetcheva. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft-ietf-manet-dsr-07.txt, February 2002. Work in progress.
- [38] John Jubin and Janet D. Tornow. The DARPA packet radio network protocols. *Proceedings of the IEEE*, 75(1):21–32, January 1987.
- [39] J. M. Kahn, R. H. Katz, and K. S. Pister. Mobile networking for smart dust. In *ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, Seattle, WA, August 1999.
- [40] Stephen Kent, Charles Lynn, Joanne Mikkelsen, and Karen Seo. Secure border gateway protocol (S-BGP) — real world performance and deployment issues. In *Symposium on Network and Distributed Systems Security (NDSS '00)*, pages 103–116, San Diego, CA, February 2000.
- [41] Minkyong Kim and Brian Noble. Mobile network estimation. In *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking (MobiCom 2001)*, pages 298–309, July 2001.
- [42] Young-Bae Ko and Nitin Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Proceedings of the Fourth International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 66–75. ACM, October 1998.
- [43] John Kohl and B. Clifford Neuman. The Kerberos Network Authentication Service (V5). RFC 1510, September 1993.
- [44] Markus G. Kuhn. Probabilistic counting of large digital signature collections. In *Proceedings of the 9th USENIX Security Symposium*, pages 73–83, August 2000.
- [45] B. Kumar. Integration of security in network routing protocols. *SIGSAC Review*, 11(2):18–25, 1993.
- [46] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.
- [47] Mark Lomas, editor. *Security Protocols—International Workshop*, volume 1189 of *Lecture Notes in Computer Science*, Cambridge, United Kingdom, April 1997. Springer-Verlag, Berlin Germany.
- [48] David A. Maltz, Josh Broch, Jorjeta Jetcheva, and David B. Johnson. The effects of on-demand behavior in routing protocols for multi-hop wireless ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1439–1453, August 1999.
- [49] David A. Maltz, Josh Broch, and David B. Johnson. Quantitative lessons from a full-scale multi-hop wireless ad hoc network testbed. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 992–997, September 2000.
- [50] Sergio Marti, T.J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehaviour in mobile ad hoc networks. In *Proceedings of the sixth annual International Conference on Mobile Computing and Networking*, pages 255–265, Boston MA, USA, August 2000.
- [51] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP selective acknowledgment options. RFC 2018, October 1996.
- [52] Alfred J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, July 1993.

- [53] R. C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology - Crypto '89*, pages 218–238, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 435.
- [54] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology - Crypto '87*, pages 369–378, Berlin, 1987. Springer-Verlag. Lecture Notes in Computer Science Volume 293.
- [55] S. Micali. Enhanced certificate revocation system. rough draft, 1995.
- [56] David L. Mills. A computer-controlled LORAN-C receiver for precision timekeeping. Technical Report 92-3-1, Department of Electrical and Computer Engineering, University of Delaware, March 1992.
- [57] David L. Mills. A precision radio clock for WWV transmissions. Technical Report 97-8-1, Department of Electrical and Computer Engineering, University of Delaware, August 1997.
- [58] Thomas Narten, Erik Nordmark, and William Allen Simpson. Neighbor Discovery for IP Version 6 (IPv6). RFC 2461, December 1998.
- [59] Panagiotis Papadimitratos and Zygmont J. Haas. Secure routing for mobile ad hoc networks. In *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, January 2002.
- [60] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of INFOCOM '97*, pages 1405–1413, April 1997.
- [61] Torben Pryds Pedersen. Electronic payments of small amounts. In Lomas [47], pages 59–68.
- [62] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *Proceedings of the SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244. ACM, August 1994.
- [63] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, pages 90–100, February 1999.
- [64] Radia Perlman. *Interconnections: Bridges and Routers*. Addison-Wesley, 1992.
- [65] Adrian Perrig, Ran Canetti, Dawn Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *Network and Distributed System Security Symposium, NDSS '01*, February 2001.
- [66] Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, May 2000.
- [67] Adrian Perrig, Yih-Chun Hu, and David B. Johnson. Wormhole protection in wireless ad hoc networks. Technical Report TR01-384, Department of Computer Science, Rice University, December 2001.
- [68] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *Seventh Annual ACM International Conference on Mobile Computing and Networks (MobiCom 2001)*, Rome, Italy, July 2001.
- [69] Raymond L. Pickholtz, Donald L. Schilling, and Laurence B. Milstein. Theory of spread spectrum communications — a tutorial. *IEEE Transactions on Communications*, 30(5):855–884, May 1982.
- [70] Guillaume Poupard and Jacques Stern. Security analysis of a practical “on the fly” authentication and signature generation. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT '98*, number 1403 in Lecture Notes in Computer Science, pages 422–436. Springer-Verlag, Berlin Germany, 1998.
- [71] Guillaume Poupard and Jacques Stern. On the fly signatures based on factoring. In *6th ACM Conference on Computer and Communications Security*, pages 37–45, November 1999.
- [72] Amir Qayyum, Laurent Viennot, and Anis Laouiti. Multipoint relaying: An efficient technique for flooding in mobile wireless networks. Technical Report Research Report RR-3898, INRIA, February 2000.
- [73] Theodore S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, New Jersey, 1996.
- [74] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4), May 1998. Special Issue on Copyright and Privacy Protection.
- [75] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.
- [76] Yakov Rekhter and Tony Li. A border gateway protocol 4 (BGP-4). RFC 1771, March 1995.
- [77] Ronald L. Rivest and Adi Shamir. PayWord and MicroMint: Two simple micropayment schemes. In Lomas [47], pages 69 – 88.
- [78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

- [79] Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *6th ACM Conference on Computer and Communications Security*, November 1999.
- [80] Claus P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.
- [81] Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO '2001*, number 2139 in Lecture Notes in Computer Science, pages 355–367. Springer-Verlag, Berlin Germany, 2001.
- [82] Bradley R. Smith and J.J. Garcia-Luna-Aceves. Securing the border gateway routing protocol. In *Global Internet'96*, London, UK, November 1996.
- [83] Bradley R. Smith, Shree Murthy, and J.J. Garcia-Luna-Aceves. Securing distance vector routing protocols. In *Symposium on Network and Distributed Systems Security (NDSS '97)*, San Diego, California, February 1997.
- [84] Frank Stajano and Ross Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In B. Christianson, B. Crispo, and M. Roe, editors, *Security Protocols, 7th International Workshop*. Springer Verlag, 1999.
- [85] Trimble Navigation Limited. Data sheet and specifications for Trimble Thunderbolt GPS Disciplined Clock. Sunnyvale, California. Available at <http://www.trimble.com/thunderbolt.html>.
- [86] Aristotelis Tsirigos and Zygmunt J. Haas. Multipath routing in mobile ad hoc networks or how to route in the presence of topological changes. In *Proceedings of IEEE MILCOM 2001*, October 2001.
- [87] Seung Yi, Prasad Naldurg, and Robin Kravets. Security-aware ad hoc routing for wireless networks. Technical Report UIUCDCS-R-2001-2241, Department of Computer Science, University of Illinois at Urbana-Champaign, August 2001.
- [88] Manel Guerrero Zapata. Secure Ad hoc On-Demand Distance Vector (SAODV) routing. IETF MANET Mailing List, Message-ID: 3BC17B40.BBF52E09@nokia.com, Available at <ftp://manet.itd.nrl.navy.mil/pub/manet/2001-10.mail>, October 8, 2001.
- [89] Kan Zhang. Efficient protocols for signing routing messages. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS '98)*, San Diego, California, March 1998. Internet Society.
- [90] Lidong Zhou and Zygmunt J. Haas. Securing ad hoc networks. *IEEE Network Magazine*, 13(6):24–30, November/December 1999.