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

Yih-Chun Hu
Carnegie Mellon University
yihchun@cs.cmu.edu

Adrian Perrig
Carnegie Mellon University
perrig@cmu.edu

David B. Johnson
Rice University
dbj@cs.rice.edu

December 16, 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 setting, assuming a 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. Ariadne prevents attackers or compromised nodes from tampering with uncompromised routes consisting of uncompromised nodes, and also prevents many types of Denial-of-Service attacks. In addition, Ariadne is efficient, using only highly efficient *symmetric* cryptographic primitives.

1. Introduction

An *ad hoc network* is a group of wireless mobile computers (or nodes), in which nodes cooperate by forwarding packets for each other to allow them to communicate beyond direct wireless transmission range. Ad hoc networks require no centralized administration or fixed network infrastructure such as base stations or access points, and can be quickly and inexpensively set up as needed. They can be used in scenarios in which no infrastructure exists, or in which the existing infrastructure does not meet application requirements for reasons such as security or cost. Applications such as military exercises, disaster relief, and mine site operation, for example, 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 [53] 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 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. On-demand routing protocols have been demonstrated to perform better with significantly lower overheads than periodic (or proactive) routing protocols in many situations [8, 27, 37, 45], since the protocol is able to react quickly to the many changes that may occur in node connectivity, yet is able to reduce (or eliminate) routing overhead in periods or areas of the network in which changes are less frequent.

In this paper, we make two contributions to the area of secure routing protocols for ad hoc networks. First, we give a model for the types of attacks possible in such a system, and we describe several new attacks on ad hoc network routing protocols. Second, 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

This work was supported in part by NSF under grant CCR-0209204, by NASA under grant NAG3-2534, by the United States Postal Service under contract USPS 102592-01-Z-0236, by DARPA under contract N66001-99-2-8913, and by gifts from Schlumberger and Bosch. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, NASA, USPS, DARPA, Schlumberger, Bosch, Rice University, Carnegie Mellon University, or the U.S. Government or any of its agencies.

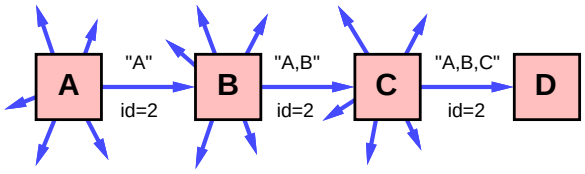


Figure 1: Example of DSR Route Discovery, in which initiator node A is attempting to discover a route to target node D.

symmetric cryptography. Relative to previous work in securing ad hoc network routing protocols, Ariadne is more secure, more effi cient, or more general (e.g., Ariadne does not require trusted hardware and does not require powerful processors).

Ariadne can authenticate routing messages using one of three schemes: shared secret keys between all pairs of nodes, shared secret keys between communicating nodes combined with broadcast authentication, or digital signatures. We primarily discuss here the use of Ariadne with TESLA [47, 48], an effi cient broadcast authentication scheme that requires loose time synchronization. Using pairwise shared keys avoids the need for synchronization, but at the cost of higher key setup overhead; broadcast authentication such as TESLA also allows some additional protocol optimizations.

In Section 2 of this paper, we summarize the basic operation of the Dynamic Source Routing protocol (DSR) [28, 29, 30], on which we base the design of our new secure routing protocol, Ariadne, and in Section 3, we review the TESLA broadcast authentication protocol that we use in Ariadne. In Section 4, we describe 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. In Section 6, we present the design of Ariadne, and in Section 7, we give an initial simulation-based performance evaluation of a basic form of Ariadne. In Section 8, we discuss related work, and in Section 9, we present our conclusions.

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) [28, 29, 30], since DSR operates *entirely* on-demand and has been well studied through both simulation and real testbed implementation [8, 27, 37, 38]. Unlike periodic protocols (e.g., [4, 44, 51]), 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 new route is needed to deliver a packet 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 benefi t of reducing the number of routing packets that need to be authenticated, thereby reducing the computational overhead needed for security. The operation of DSR is divided into two activities: *Route Discovery* and *Route Maintenance*. In this section, we describe the basic form of Route Discovery and Route Maintenance in 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 Route Discovery to fi nd a route; this node is known as the *initiator* of the Route Discovery, and the destination of the packet is known as the Discovery's *target*, as illustrated in Figure 1. The initiator (node A) transmits a ROUTE REQUEST packet as a local broadcast, specifying the target (node D) and a unique identifi er from the initiator A. Each node receiving the ROUTE REQUEST, if it has recently seen this request identifi er from the initiator or if its own address is already present in an address list in the REQUEST, discards the REQUEST. Otherwise, the appends its own node address to the address 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 accumulated 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 specifi ed route to some destination detects if that route has broken, for example because two nodes in it have moved too far apart; an example of Route Maintenance is shown in Figure 2. DSR is based on *source routing*: when sending a packet, the originator lists in the header of the packet the complete sequence of nodes through which the packet is to be forwarded. Each node

Figure 2: Example of DSR Route Maintenance, in which intermediate node C detects a broken link to D when forwarding a packet from source node A.

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, passive acknowledgment [31], or network-layer acknowledgment. If, after a limited number of local retransmissions of the packet, a node in the route is unable to make this confirmation (such as node C in Figure 2), it returns a ROUTE ERROR to the original source of the packet (node A), identifying the link from itself to the next node (node D) as broken. The sender then removes this 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.

The DSR protocol also defines a number of optimizations to these mechanisms (e.g., [19, 20, 28, 29, 30, 34]). Some of these optimizations, such as flow state [20], are relatively easy to secure (flow state requires only broadcast authentication of control messages), whereas others, such as link-state caching [19], are more difficult (link-state caching requires some mechanism to authenticate links, but Ariadne only attempts to authenticate nodes). The use of these DSR optimizations is beyond the scope of this paper; we secure here only a basic version of DSR, with a limited path cache and without these optimizations.

3. Overview of TESLA

In this paper, we describe Ariadne primarily using the TESLA [47, 48] broadcast authentication protocol for authenticating routing messages, since TESLA is efficient and adds only a single message authentication code (MAC) to a message for broadcast authentication. Adding a MAC (computed with a shared key) to a message can provide secure authentication in point-to-point communication; for broadcast communication, 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 the receivers can only verify the authentication information. TESLA differs from traditional asymmetric protocols such as RSA [54] in that TESLA achieves this asymmetry from clock synchronization and delayed key disclosure, rather than from computationally expensive one-way trapdoor functions.

To use TESLA for authentication, each sender chooses a random initial key K_N and generates a *one-way key chain* by repeatedly computing a one-way hash function H on this starting value: $K_{N-1} = H[K_N]$, $K_{N-2} = H[K_{N-1}]$, ... In general, $K_i = H[K_{i+1}] = H^{N-i}[K_N]$. To compute any previous key K_j from a key K_i, $j < i$, a node uses the equation $K_j = H^{i-j}[K_i]$. To authenticate any received value on the one-way chain, a node applies this equation to the received value to determine if the computed value matches a previous known authentic key on the chain. Coppersmith and Jakobsson present efficient mechanisms for storing and generating values of hash chains [13].

Each sender pre-determines a schedule of the time at which it publishes (or discloses) each key of its one-way key chain, in the reverse order from generation; that is, a sender publishes its keys in the order $K_0, K_1, \ldots, K_N$. A simple key disclosure schedule, for example, would be to publish key K_i at time $T_0 + i \times I$, where T_0 is the time at which K_0 is published, and I is the key publication interval.

TESLA relies on a receiver's ability to determine which keys a sender may have already published, based on loose time synchronization between nodes. Let Δ be the maximum time synchronization error between any two nodes; the value Δ must be known by all nodes. To send a packet, the sender uses a pessimistic upper bound τ on the end-to-end network delay between nodes and picks a key K_i from its one-way key chain which, at the time any receiver is expected to receive the packet, the receiver will believe has not yet been published. For example, the sender could choose a key K_i that it will not publish until a time at least $\tau + 2\Delta$ in the future; the 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. In sending the packet, the sender adds a message authentication code (MAC), computed using key K_i, to the packet. 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 published already. Since the receiver knows the sender's clock may be faster by Δ, the receiver will reject the packet unless it is received at least Δ before the scheduled key release time, so the receiver must be able to verify that the key is released at time $t_s + \tau + 2\Delta$ or later.

When a receiver receives a packet authenticated with TESLA, it first verifies the TESLA *security condition* that the key K_i used to authenticate the packet cannot yet have been published. For example, if the local packet arrival time is t_r, and the receiver knows that the earliest time at which the sender will disclose the key K_i is $T_0 + i \times I$, the receiver needs to verify only that $t_r \leq (T_0 + i \times I - \Delta)$, implying that K_i has not yet been published. Otherwise, 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 this check is successful, the receiver buffers the packet and waits for the sender to publish key K_i. When the receiver then receives K_i, it authenticates K_i as described above, and then authenticates stored packets that were authenticated with any key K_j, where $j \leq i$. TESLA remains secure even if the end-to-end delay is larger than τ, although some receivers may be required to discard the packet.

4. Assumptions

4.1. 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 using the HMAC algorithm [3].

For notational convenience we assume hash and MAC functions that take a variable number of arguments, simply concatenating them in computing the function.

4.2. Network Assumptions

The physical layer of a wireless network is often vulnerable to denial of service attacks such as jamming. Mechanisms such as spread spectrum [50] have been extensively studied as means of providing resistance to physical jamming, and we thus disregard such physical layer attacks here.

We assume that network links are bidirectional; that is, if a node A is able to receive packets transmitted directly by some node B, then B is able to receive packets transmitted directly by A. It is possible to use a network with unidirectional links if such links are detected and avoided; such detection may also otherwise be necessary, since many wireless Medium Access Control protocols require bidirectional links, as they make use of bidirectional exchange of several link-layer frames between a source and destination to help avoid collisions and improve reliability [6, 26].

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 to at least 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, and the development of secure Medium Access Control protocols is an active area of research. In this paper, we disregard attacks on Medium Access Control protocols.

We assume that the network may drop, corrupt, reorder, or duplicate packets in transmission.

When Ariadne is used with a broadcast authentication protocol, we naturally inherit all of that protocol's assumptions. For example, when TESLA is used, each node in the network must be able to estimate the end-to-end transmission time to any other node in the network; TESLA permits this value to be chosen adaptively and pessimistically. When this time is chosen to be too large, authentication delay increases, reducing protocol responsiveness; when it is chosen to be too small, authentic packets may be rejected, but security is not compromised.

4.3. Node Assumptions

The resources of different ad hoc network nodes may vary greatly, from nodes with very little computational resources, to resource-rich nodes equivalent in functionality to high-performance workstations. To make our results as general as possible, we have designed Ariadne to support nodes with few resources, such as Palm PDAs or RIM pagers.

Most previous work on secure ad hoc network routing relies on *asymmetric* cryptography such as digital signatures [63, 65]. However, computing such signatures on resource-constrained nodes is expensive, and we assume that 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 [9]; on a Palm Pilot PDA (16 MHz Motorola 68000 "Dragonball" processor) or RIM pager (10 MHz custom Intel 386 processor), a 512-bit RSA [54] signature generation takes 2.4–5.8 seconds, and signature verifi cation takes 0.1–0.6 seconds, depending on the public exponent.

When Ariadne is used with TESLA for broadcast authentication, we assume that all nodes have loosely synchronized clocks, such that the difference between any two nodes' clocks does not exceed Δ; the value of Δ must be known by all nodes in the network. Accurate time synchronization can be maintained, for example, with off-the-shelf hardware based on GPS [12, 60], although the time synchronization signal itself may be subject to attack [15]. We assume that nodes compensate clock drift with periodic re-synchronization. Microcomputer-compensated crystal oscillators [5] can provide sub-second accuracy for several months; if normal crystal oscillators are used, the value of Δ can be chosen to be as large as necessary, though a corresponding reduction in protocol responsiveness will result.

We do *not* assume trusted hardware such as tamperproof modules. Secure routing with trusted hardware is much simpler, since node compromise is assumed to be impossible (Section 6.1).

4.4. Security Assumptions and Key Setup

The security of Ariadne relies on the secrecy and authenticity of keys stored in nodes. Ariadne relies on the following keys to be set up, depending on which authentication mechanism is used:

- If *pairwise shared secret keys* are used, we assume a mechanism to set up the necessary $n(n+1)/2$ keys, if n is the number of nodes in the network.
- If *TESLA* is used, we assume a mechanism to set up shared secret keys between pairs of nodes that communicate, and to distribute one authentic public TESLA key for each node.
- If *digital signatures* are used, we assume a mechanism to distribute one authentic public key for each node.

To set up shared secret keys, we can use a variety of mechanisms. For example, a key distribution center may be used, which shares a secret key with each node and sets up shared secret keys with communicating nodes, such as in Kerberos [35] or SPINS [49]; shared secret keys can be bootstrapped from a Public Key Infrastructure (PKI) using protocols such as TLS [14]; or shared secret keys can be pre-loaded at initialization, possibly through physical contact [59]. Menezes et al discuss several key setup protocols [41].

To set up authentic public keys, we can either embed all public keys at initialization in each node, or assume a PKI and embed the trusted Certifi cation Authority's public key in each node and then use that key to authenticate the public keys of other nodes. Another approach proposed by Hubaux et al [25] bootstraps trust relationships based on PGP-like certifi cates.

Ariadne also requires that each node have an authentic element from the Route Discovery chain (Section 6.5) of every node initiating Route Discoveries. These keys can be set up in the same way as a public key.

Key setup is an expensive operation. Setting up shared secret keys requires authenticity and confi dentiality, whereas setting up public keys only requires authenticity. Furthermore, fewer public keys are generally needed, because in a network with n nodes, only n public keys are needed and can potentially be broadcast, whereas $n(n+1)/2$ secret keys need to be set up in the case of pairwise shared secret keys. In Section 4.5, we describe a mechanism to set up these keys without relying on Ariadne, thus avoiding the circular dependency between key setup and the routing protocol.

4.5. Establishing Authenticated Keys using a Trusted 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 4.4. 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 this type of key setup can be done, if desired, with no established routing within Ariadne.

As described in Section 4.1, we assume that each node A shares secret MAC keys K_{AT} and K_{TA} between itself and the trusted KDC T for use with a message authentication code (MAC) algorithm such as HMAC [3] (one key for each direction of communication). For use with this method of establishing authenticated keys using a trusted KDC, we further assume that each node A similarly shares secret encryption keys K'_{AT} and K'_{TA} between itself and the KDC T. These shared keys K_{AT}, K'_{AT}, K_{TA}, and K'_{TA} for node A could, for example, all be derived by A and the KDC T using a single shared a master secret key $\mathcal{X}_{AT} = \mathcal{X}_{TA}$ (no direction information is associated with this key) using a Pseudo-Random Function (PRF) [16]: if F is a PRF, then let $K_{AT} = F_{\mathcal{X}_{AT}}(1)$, $K'_{AT} = F_{\mathcal{X}_{AT}}(2)$, $K_{TA} = F_{\mathcal{X}_{AT}}(3)$, and $K'_{TA} = F_{\mathcal{X}_{AT}}(4)$.

We also assume here that each node can obtain an authentic TESLA key of the KDC; that the KDC has an authentic TESLA key for each other node; and if MW-Chains [23] are used to thwart ROUTE REQUEST floods as described in Section 6.3, that each node can obtain an authentic MW-Chain head for the KDC.

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 by each node as in Ariadne, except that each node receiving this special REQUEST for the first time also returns a ROUTE REPLY. The KDC can then use each returned route to send authenticated keys to each node in the network. Alternatively, this special Route Discovery procedure can be repeated periodically; when a node needs a shared key with some other node, it waits until the next such special Discovery is initiated by the KDC node, and at that time includes as part of its ROUTE REPLY to the KDC the list of nodes for which it needs authenticated keys.

5. Ad Hoc Network Routing Security

In this section, we define a taxonomy of types of attackers 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 two main attacker classes, *passive* and *active*. The passive attacker does not send messages; it only eavesdrops on the network. Passive attackers are mainly threats against the privacy or anonymity of communication, rather than against the functioning of the network or its routing protocol, and thus we do not discuss them further here.

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

Our protocol does not require a trusted Key Distribution Center (KDC) in the network, but some ad hoc networks may use one for key setup, as mentioned in Section 4.4. 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 on Ad Hoc Network Routing Protocols

Attacks on ad hoc network routing protocols 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 cause all packets for some destination to be routed to itself and could 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 [39], 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 [24], using a pair of attacker nodes A and B linked via a private network connection. Every packet (or selected packets) that 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.

The *rushing* attack is a malicious attack that is targeted against on-demand routing protocols that find routes through a Route Discovery protocol and use duplicate suppression of the ROUTE REQUEST messages in that protocol at each node [22]. An attacker disseminates ROUTE REQUESTs quickly throughout the network, suppressing any later legitimate ROUTE REQUESTs when nodes drop them due to the duplicate suppression.

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 Active-VC attacker can try to extract maximum resources from the nodes on both sides of the vertex cut; for example, it might forward 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 routing loops, and if a maximum route length can be enforced, then an attacker that can inject extra data packets has limited attack power. In particular, if routes are limited to ν hops, then each data packet transmitted by the attacker causes only at most a fixed number of additional transmissions; more generally, if at most one control packet can be sent in response to each data packet (e.g., a ROUTE ERROR), and that control packet is limited to ν hops, then an individual data packet can cause only 2ν individual transmissions. We consider an attack a DoS attack only if 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. An example of a DoS attack is for the attacker to send a single packet that results in a packet flood throughout the network.

6. Ariadne

6.1. Design Goals

We aim for resilience against Active-1-x and Active-y-x attackers. Ideally, the probability that the routing protocol delivers messages degrades gracefully when nodes fail or are compromised. Our goal is to design simple and efficient mechanisms achieving high attack robustness. These mechanisms should be sufficiently general to allow application to a wide range of routing protocols.

Defending against an Active-0-x attacker is relatively easy. A network-wide shared secret key limits the attacker to replaying messages. Thus the main attacks remaining are the wormhole and rushing attacks (Section 5.2). *Packet leashes* [24] can prevent both attacks because they prevent an Active-0-x attacker from retransmitting packets. These approaches also trivially secure a network routing protocol that uses tamperproof hardware, since the strongest attacker in such an environment is an Active-0-x attacker, assuming all routing and security functionality (including packet leashes) is implemented in the secure hardware.

Most routing disruption attacks we present in Section 5.2 are caused by malicious injection or altering of routing data. To prevent these attacks, each node that interprets routing information must verify the origin and integrity of that data; that is, it must authenticate the data. Ideally, the initiator of the Route Discovery can verify the origin of each individual data field in the ROUTE REPLY.

We need an authentication mechanism with low computation and communication overhead. An inefficient authentication mechanism could be exploited by an attacker to perform a Denial-of-Service (DoS) attack by flooding nodes with malicious messages, overwhelming them with the cost of verifying authentication. Thus, for point-to-point authentication of a message, we use a message authentication code (MAC) (e.g., HMAC [3]) and a shared key between the two parties. However, setting up the shared keys between the initiator and all the nodes on the path to the target may be expensive. We thus also propose using the TESLA broadcast authentication protocol (Section 3) for authentication of nodes on the routing path. However, we also discuss MAC authentication with pairwise shared keys,

for networks capable of inexpensive key setup, and we discuss digital signatures for authentication, for networks with extremely powerful nodes.

As a general design principle, a node trusts only itself for acquiring information about which nodes in the network are malicious. This approach helps avoid blackmail attacks, where an attacker constructs information to make a legitimate node appear malicious. In our design, we assume that a sender trusts the destination with which it communicates, for authenticating nodes on the path between them. This assumption is straightforward, as the destination node can control all communication with the sender anyway. However, the destination node can potentially blackmail nodes on the path to the sender. The sender thus needs to keep a separate blacklist for each destination.

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.

Our protocol does not prevent an attacker from injecting data packets. As we described in Section 5.2, injecting a packet results in a DoS attack only if it floods the network. Since data packets cannot flood the network, we do not explicitly protect against packet injection. However, malicious ROUTE REQUEST messages that flood the network do classify as a DoS attack, and we thus prevent this attack with a separate mechanism that we describe in Section 6.5.

6.2. Basic Ariadne Route Discovery

In this section, we describe the basic operation of Route Discovery in Ariadne. We first overview the features of the protocol in three stages: we present a mechanism that enables the target of a Route Discovery to verify the authenticity of the ROUTE REQUEST; we then present three alternative mechanisms for authenticating data in ROUTE REQUESTs and ROUTE REPLYs; and we present an efficient per-hop hashing technique to verify that no node is missing from the node list in the REQUEST. After this overview, we present in detail the operation of Route Discovery in Araidne when TESLA is used as the authentication mechanism. In the following discussion, we assume that some initiator node S performs a Route Discovery for a target node D, and that they share the secret keys K_{SD} and K_{DS}, respectively, for message authentication in each direction.

Target Authenticates ROUTE REQUESTs. To convince the target of the legitimacy of each field in a ROUTE REQUEST, the initiator simply includes in the REQUEST a MAC computed with key K_{SD} over unique data, for example a timestamp. The target can easily verify the authenticity and freshness of the ROUTE REQUEST using the shared key K_{SD}.

Three Techniques for Route Data Authentication. In a Route Discovery, the initiator wants to authenticate each individual node in the node list of the ROUTE REPLY. A secondary requirement is that the target can authenticate each node in the node list of the ROUTE REQUEST, so that it will return a ROUTE REPLY only along paths that contain only legitimate nodes. In this section, we present three alternative techniques to achieve node list authentication: the TESLA protocol, digital signatures, and standard MACs.

When Ariadne Route Discovery is used with *TESLA*, each hop authenticates the new information in the REQUEST. The target buffers and does not send the REPLY until intermediate nodes can release the corresponding TESLA keys. The TESLA security condition is verified at the target, and the target includes a MAC in the REPLY to certify that the security condition was met. TESLA requires each packet sender to choose a a pessimistic upper bound τ on the end-to-end network delay between nodes for sending this packet, in order to select the TESLA key it will use to authenticate it. Choices of τ do not affect the security of the protocol, although values that are too small may cause the Route Discovery to fail. Ariadne can choose τ adaptively, by increasing τ when a Discovery fails. In addition, the target of the Discovery could provide feedback in the ROUTE REPLY when τ was chosen too large.

If Ariadne Route Discovery is used with *digital signatures*, instead, the authentication differs in that no Route Discovery chain element is required (Section 6.5). In addition, the MAC list in the ROUTE REQUEST (described below) becomes a signature list, where the data used to compute the MAC is instead used to compute a signature. Rather than computing the target MAC using a Message Authentication Code, a signature is used. Finally, no key list (also described below) is required in the REPLY.

Ariadne Route Discovery using *MAC*s is the most efficient of the three alternative authentication mechanisms, but it requires pairwise shared keys between all nodes. When Ariadne is used in this way, the MAC list in the ROUTE REQUEST is computed using a key shared between the target and the current node, rather than using the TESLA key of the current node. The MACs are verified at the target and are not returned in the ROUTE REPLY. As a result, the target MAC is not computed over the MAC list in the REQUEST. In addition, no key list is required in the REPLY.

Per-Hop Hashing. Authentication of data in routing messages is not sufficient, as an attacker could remove a node from the node list in a ROUTE REQUEST. We use one-way hash functions to verify that no hop was omitted, and we call this approach *per-hop hashing*. To change or remove a previous hop, an attacker must either hear a ROUTE REQUEST without that node listed, or it must be able to invert the one-way hash function.

Ariadne Route Discovery with TESLA. We now describe in detail the version of Ariadne Route Discovery using TESLA broadcast authentication. 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 a ROUTE REPLY. To secure the ROUTE REQUEST packet, Ariadne 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: ⟨ROUTE REQUEST, *initiator, target, id, time interval, hash chain, node list, MAC list*⟩. 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 time, the time interval could be set to any time interval for which the key is not released within the next $\tau + 2\Delta$ time. The initiator of the REQUEST then initializes the *hash chain* to $\text{MAC}_{K_{SD}}(\textit{initiator, target, id, time interval})$ 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 ⟨*initiator, id*⟩ 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

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

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: ⟨ROUTE REPLY, *target, initiator, time interval, node list, MAC list, target MAC, key list*⟩. 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; it then 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 3 shows an example of Route Discovery in Ariadne.

6.3. Basic Ariadne Route Maintenance

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

$$
\begin{array}{ll}
S: & h_0 = \text{MAC}_{K_{SD}}(\text{REQUEST}, S, D, id, ti) \\
S \rightarrow *: & \langle \text{REQUEST}, S, D, id, ti, h_0, (), () \rangle \\
 & \\
A: & h_1 = H[A, h_0] \\
 & M_A = \text{MAC}_{K_{A_{ti}}}(\text{REQUEST}, S, D, id, ti, h_1, (A), ()) \\
A \rightarrow *: & \langle \text{REQUEST}, S, D, id, ti, \underline{\boldsymbol{h_1}}, (\underline{\boldsymbol{A}}), (\underline{\boldsymbol{M_A}}) \rangle \\
 & \\
B: & h_2 = H[B, h_1] \\
 & M_B = \text{MAC}_{K_{B_{ti}}}(\text{REQUEST}, S, D, id, ti, h_2, (A, B), (M_A)) \\
B \rightarrow *: & \langle \text{REQUEST}, S, D, id, ti, \underline{\boldsymbol{h_2}}, (A, \underline{\boldsymbol{B}}), (M_A, \underline{\boldsymbol{M_B}}) \rangle \\
 & \\
C: & h_3 = H[C, h_2] \\
 & M_C = \text{MAC}_{K_{C_{ti}}}(\text{REQUEST}, S, D, id, ti, h_3, (A, B, C), (M_A, M_B)) \\
C \rightarrow *: & \langle \text{REQUEST}, S, D, id, ti, \underline{\boldsymbol{h_3}}, (A, B, \underline{\boldsymbol{C}}), (M_A, M_B, \underline{\boldsymbol{M_C}}) \rangle \\
 & \\
D: & M_D = \text{MAC}_{K_{DS}}(\text{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C)) \\
D \rightarrow C: & \langle \text{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), \underline{\boldsymbol{M_D}}, () \rangle \\
 & \\
C \rightarrow B: & \langle \text{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (\underline{\boldsymbol{K_{C_{ti}}}}) \rangle \\
 & \\
B \rightarrow A: & \langle \text{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (K_{C_{ti}}, \underline{\boldsymbol{K_{B_{ti}}}}) \rangle \\
 & \\
A \rightarrow S: & \langle \text{REPLY}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (K_{C_{ti}}, K_{B_{ti}}, \underline{\boldsymbol{K_{A_{ti}}}}) \rangle
\end{array}
$$

Figure 3: 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.

to the next hop after a limited number of retransmission attempts. In this section, we discuss mechanisms for securing ROUTE ERRORs, but we do not consider the case of attackers not sending ERRORs (Section 6.4).

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

When using broadcast authentication, such as TESLA, a ROUTE ERROR packet in Ariadne contains six fields: ⟨ROUTE ERROR, *sending address*, *receiving address*, *time interval*, *error MAC*, *recent TESLA key*⟩. 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. For example, if 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; the the *sending address* in this example is set to B, and the *receiving address* is 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 ERROR. We use TESLA for authenticating ROUTE ERRORs so that forwarding nodes can also authenticate and process the ROUTE 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 ROUTE 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 ⟨*sending address*, *receiving address*⟩ 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 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 sender stops using that route, there will be no need to complete the authentication of the 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 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 following technique is quite effective: each node keeps in memory a table containing the information from each ROUTE ERROR awaiting authentication. We manage this table such that the probability that the information from an ERROR is in the table is independent of the time that this node received that ROUTE 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.

When digital signatures or pairwise shared keys are used, this memory consumption attack is not possible, and the authentication is more straightforward. A ROUTE ERROR need not include a *time interval* or *recent TESLA key*. Furthermore, the *error MAC* is changed to a digital signature when digital signatures are used. When pairwise shared keys are used, the *error MAC* is computed based on the key shared between the original sender of the packet and the sender of the ROUTE ERROR, rather than on the TESLA key of the sender of the ERROR.

6.4. Thwarting Effects of Routing Misbehavior

The protocol described so far is vulnerable to an Active-1-1 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 [39] 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 [40]; this feedback approach is somewhat similar that used in IPv6 for Neighbor Unreachability Detection [42]. 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 DoS 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. A packet sent for this purpose can be a normal data packet, or, if all packets are secured using end-to-end encryption, a padded "probe" packet can be used.

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 [61] also discuss the use of multiple routes for increasing reliability, although they do not discuss this technique with respect to secure routing protocols.

Malicious nodes can also be avoided during Route Discovery. Each ROUTE REQUEST can include a list of nodes to avoid, and the MAC that forms the initial hash chain element (h_0) is then also computed over that list of nodes.

Malicious nodes cannot add or remove nodes from this list without being detected by the target. Choosing which nodes to avoid in this way is beyond the scope of this paper.

6.5. 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 (Sections 6.2 and 6.3), a ROUTE REQUEST is not authenticated until it reaches its target, thus allowing an Active-1-1 attacker to cause such network-wide floods. (An Active-0-1 can be thwarted by using a network-wide authentication key, as described in Section 7.2.)

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 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 computationally slightly more expensive, but it 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.

Route Discovery chains can also be constructed from a chain-based one-time signature, such as the Merkle-Winternitz construction [23, 55, 64]. The chain can then be used to sign any set of immutable fields in the initial ROUTE REQUEST, and the signature distributed with the REQUEST. In our design, the only immutable field is the target address, since the identifier is the chain element used for the current Route Discovery, and the time interval can also be derived from that chain element. As a result, the one-time signature scheme needs to sign very few bits, and steps in the Route Discovery chain can be very inexpensive. For example, in a network with 50 nodes, it suffices to represent 49 possible targets (since the initiator is never the target). If the Merkle-Winternitz construction is used with two signature chains of length 7 and a checksum chain of length 13, each REQUEST is just 20 bytes longer, and one step in the hash chain costs just 27 hash operations. If each node is permitted to initiate one Route Discovery per second, the amortized cost of using Merkle-Winternitz chains in this network is just 1350 hash operations per second.

6.6. Optimizations for Ariadne

Caching Improvements. When Ariadne is used with broadcast authentication such as TESLA, additional route caching is possible. In the basic Route Discovery mechanism described in Section 6.2, only the initiator of the Discovery can use the route in the REPLY, since the *target MAC* field of the REPLY can only be verified by the initiator. However, if the appropriate data is also broadcast authenticated, any node along a path returned in a REPLY can use that route to reach the target. For example, if TESLA is used as the broadcast authentication protocol, a *target authenticator* is placed the packet in addition to the *target MAC*, and is computed using a TESLA key that is not expected to be disclosed until Δ after the last REPLY reaches the initiator (where Δ is the maximum time difference between two nodes). That TESLA key is then disclosed, after appropriate delay, by sending it to the initiator along each path traversed by a REPLY.

Reduced Overhead. When Ariadne is used with symmetric authentication (such as TESLA or pairwise shared keys), some fields can be calculated by the receiver rather than included in the packet [23]. In particular, the MAC list in both the REQUEST and REPLY can be eliminated, and h_i can be computed using

$$MAC_{K_{A_i}}(\text{REQUEST}, S, D, id, ti, h_{i-1}, (A_1, \ldots, A_i)) \quad .$$

The verifier (initiator with delayed broadcast authentication, and target with pairwise shared keys) can then recompute each h_i given the disclosed (or known) symmetric keys.

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

7. Ariadne Evaluation

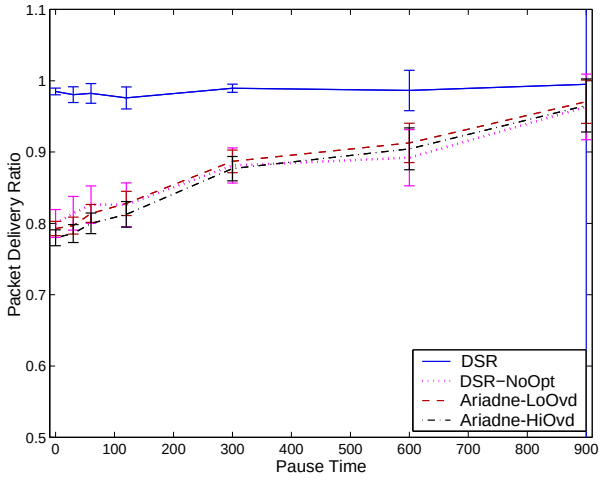
7.1. Simulation-Based Performance Evaluation

To evaluate the Ariadne without attackers, we used the *ns-2* simulator, with our mobility extensions [8]. The *ns-2* simulator has been used extensively in evaluating the performance of ad hoc network routing protocols. These simulations model radio propagation using the realistic *two-ray ground reflection* model [52] 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) [26]. The parameters used for our simulation are given in Table 1.

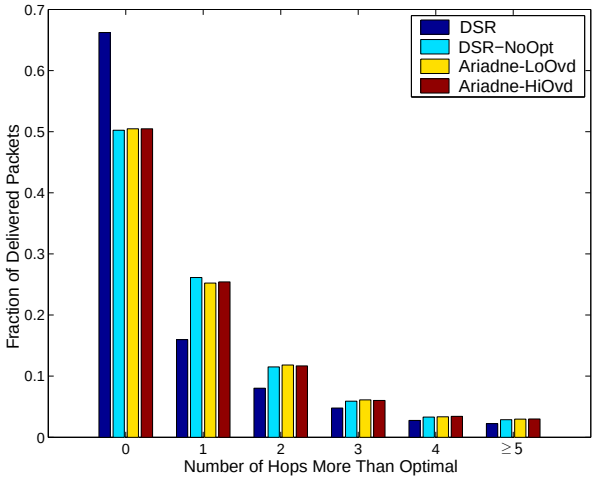
We evaluated the version of Ariadne that uses TESLA for broadcast authentication and shared keys only between communicating nodes. We also simulate the effect of adding the Reduced Overhead optimization described in Section 6.6; we refer to the unoptimized version as "Ariadne-HiOvd" and the optimized version as "Ariadne-LoOvd."

We modeled these versions of Ariadne by modifying our *ns-2* DSR model 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 this version of Ariadne versus the current version of DSR [19], which we call simply "DSR," and with an unoptimized version of DSR, which we call "DSR-NoOpt." In DSR-NoOpt, 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.

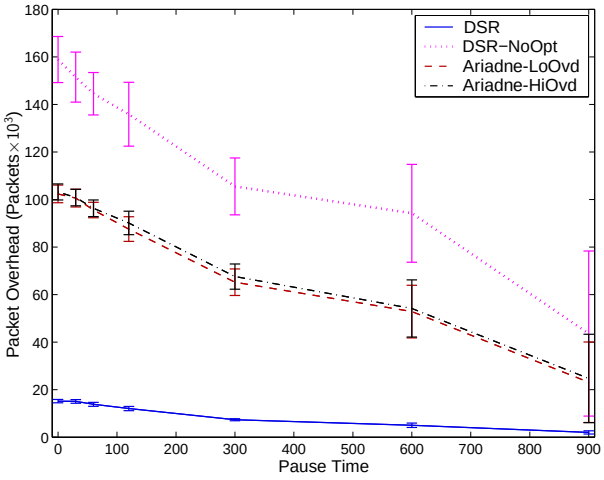
Each node in our simulation moves according to the *random waypoint* model [29]: a node starts at a random position, waits for a duration called the *pause time*, and then chooses a new random location and moves there with a velocity uniformly chosen between 0 and v_{max}. When it arrives, it waits for the pause time and repeats the process. Like much previous work in evaluating ad hoc network routing protocols (e.g., [8, 19, 27]), 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 in this respect. All protocols were run on identical movement and communication scenarios. We computed six metrics for each simulation run:



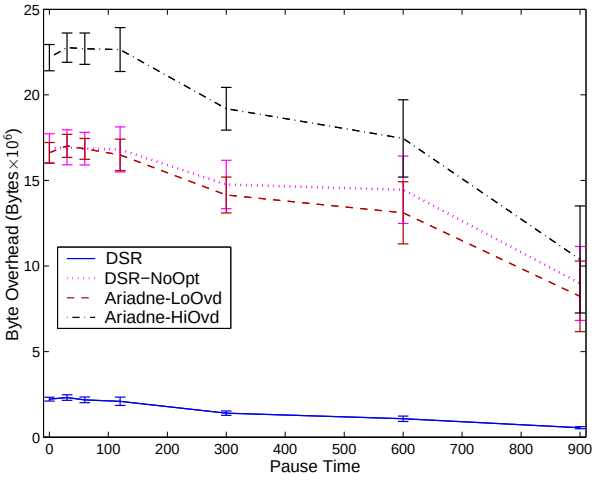
(a) Packet Delivery Ratio



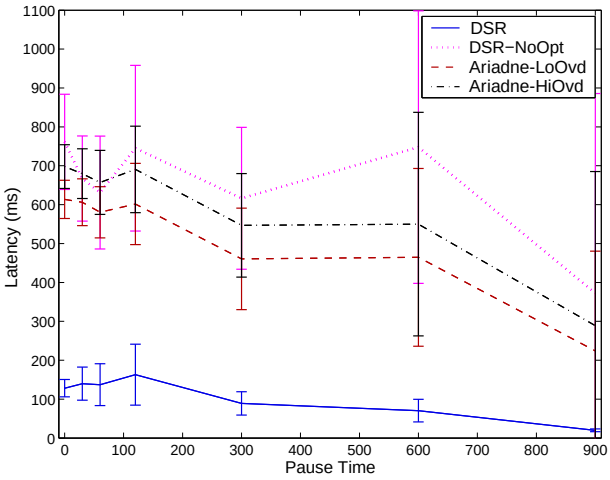
(b) Path Optimality



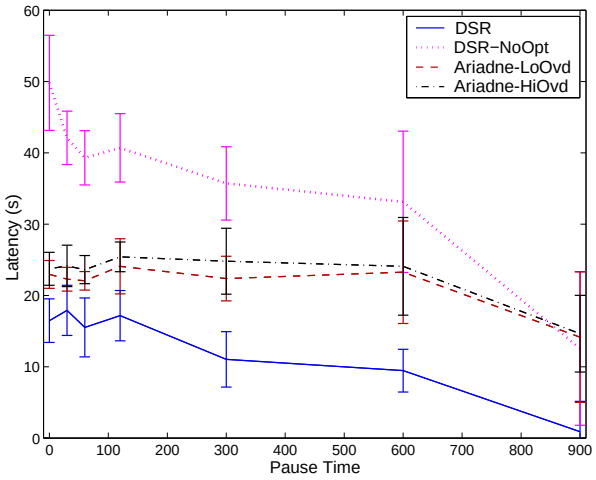
(c) Packet Overhead



(d) Byte Overhead



(e) Average Latency



(f) 99.99th Percentile Latency

Figure 4: Performance results comparing Ariadne with the standard DSR protocol and with a version of DSR with all DSR optimizations not present in Ariadne disabled. Results are based on simulation over 50 runs, and the error bars represent the 95% confi dence interval of the mean.

- *Packet Delivery Ratio (PDR)*: The fraction of application-level data packets sent that are actually received at the respective 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, counting each hop as above.
- *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 (minimum possible) hop length as determined by an off-line omniscient algorithm.

Figure 4(a) shows the Packet Delivery Ratio (PDR) for each protocol. Removing the optimizations from DSR to produce DSR-NoOpt reduces PDR by an average of 12.7%; adding Ariadne-HiOvd security further reduces PDR by just an additional 1.12% on average, and does not reduce PDR by more than an additional 2.8% at any pause time. Ariadne with reduced overhead recovers most of this PDR loss; it has PDR just 0.015% lower (on average) than DSR-NoOpt. Ariadne with reduced overhead outperforms DSR-NoOpt at low mobility, but has lower PDR at high mobility.

Surprisingly, Ariadne outperforms DSR-NoOpt at lower levels of mobility. This improved performance results from the average half-second delay (one half 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-NoOpt 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-NoOpt.

From examining PDR across these protocols, we draw three conclusions. First, Ariadne Route Discovery operates more slowly but also finds better routes, since those routes must exist for long enough to return a ROUTE REPLY. Second, these better routes offset the disadvantage that Ariadne ROUTE ERRORs cannot be processed until the TESLA key used is disclosed, causing 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. Finally, the increased overhead of Ariadne-HiOvd is almost fully responsible for its lower PDR relative to DSR-NoOpt.

Figure 4(b) shows Path Optimality. In DSR, the average number of hops along a route used by a packet is 0.700 hops more than the minimum possible, based on the nominal wireless transmission range of 250 m per hop. In DSR-NoOpt, routes used are on average 0.264 hops longer than in DSR, and in both versions of Ariadne, routes used average 0.011 hops longer than in DSR-NoOpt. DSR-NoOpt performs slightly better than Ariadne because it initiates more Route Discoveries and thus tends to more quickly find shorter routes when they become available than does Ariadne.

Figures 4(c) and 4(d) show the packet and byte overhead, respectively. Ariadne has *consistently lower* packet overhead than DSR-NoOpt, because Ariadne tends to find more stable routes than DSR-NoOpt, 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. Byte overhead in Ariadne-HiOvd is significantly worse than in either DSR or DSR-NoOpt, due to the authentication overhead in ROUTE REQUEST, REPLY, and ERROR packets. Suprisingly, Ariadne-LoOvd has lower overhead than DSR-NoOpt, because the extra packets sent by DSR-NoOpt outweigh the additional authentication information in each Ariadne-LoOvd routing control packet.

Figures 4(e) and 4(f) show the average and 99.99th percentile latency for the protocols, respectively. Because of the reduced number of broken links that Ariadne uses relative to DSR-NoOpt, Ariadne generally has better latency than DSR-NoOpt. Ariadne-LoOvd shows significantly lower mean latency than Ariadne-HiOvd, because it has lower byte overhead and thus creates less congestion. The difference is also present in the 99.99th percentile latency, but there, channel contention is a smaller fraction of the overall latency.

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.

Intuitively, Ariadne Route Discovery is successful when at least one of the REPLYs returned by the target is a working route. Since the target of a Route Discovery returns a route for each of its neighbors, if the first REQUEST from a particular Discovery to reach any neighbor of the target has passed through no malicious nodes, that Discovery will succeed.

To more formally 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 the minimum latency path between the initiator of the Discovery and that neighbor is uncompromised, 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 may route packets along one such uncompromised route.

To argue for the correctness of the first property, we note that if the minimum latency path between the initiator and a neighbor of the destination is 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.

Although it may not be possible to achieve reliable broadcast securely or efficiently, we assume that most broadcast packets are received, and hence the properties listed above generally hold.

We now consider Ariadne using our taxonomy of attacks that we present in Section 5.1. We list different attacker configurations in increasing strength, and discuss how Ariadne resists some possible attacks. Ariadne resists many more attacks, but only a representative sample are discussed here.

Since Ariadne does not attempt to provide anonymous routing, passive attackers can eavesdrop on all routing traffic sent by nodes within range of those attackers. They can also perform traffic analysis on any packets sent or forwarded by nodes within range of the attackers.

When replay protection and a global MAC key are used, an Active-0-x attacker (for $x \geq 1$) can at most perform wormhole and rushing attacks. Packet leashes can prevent these attacks [24].

An Active-1-1 attacker may attempt the following attacks:

- Create a gray hole or black hole by removing nodes in a ROUTE REQUEST; however, the per-hop hash mechanism 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.
- Create routing loops. Intuitively, the use of source routes prevents loops, since a packet passing through only legitimate nodes will not be forwarded into a loop. An attacker can create a routing loop by modifying the source route each time around the loop; 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.
- Flood network with many ROUTE REQUESTs. Since the source address of each 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. An upper bound on the sending rate can be enforced either by rate limiting of REQUESTs at each node or synchronizing Route Discovery chain elements with time (Section 6.5).
- Perform a rushing attack (Section 5.2). Rushing attacks can be probabilistically prevented by slightly modifying the Route Discovery protocol [22].

Multiple attackers that have compromised one node (Active-1-x, for $x > 1$) may attempt to construct a wormhole, but append the address and key of the compromised node in each REQUEST forwarded across this wormhole. Packet leashes alone cannot prevent this attack, but packet leashes and GPS can be used in conjunction to ensure that an Active-1-x wormhole attack can be no worse than an Active-1-1 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 Active-y-x attacker configuration, where $1 < y \leq x$) 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 Active-y-x 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 (an Active-VC 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 Active-VC 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 [46] proposed *flooding NPBR*, an on-demand protocol designed for wired networks that floods each packet through the network. Flooding NPBR allocates a fraction of the bandwidth along each link to each node, and uses digital signatures to authenticate all packets. Unfortunately, this protocol has high overhead in terms of the computational resources necessary for digital signature verification and in terms of its bandwidth requirements. Furthermore, estimating and guaranteeing available bandwidth in a wireless environment is difficult [33].

Other wired network protocols have secured periodic routing protocols with *asymmetric* cryptography, such as Kent et al [32], Perlman's *link-state NPBR*, Kumar's secure link-state protocol [36], and Smith et al [57, 58]. However, nodes in an ad hoc network may not have sufficient resources to verify an asymmetric signature; in particular, an attacker can trivially flood a victim with packets containing invalid signatures, but verification can be prohibitively expensive for the victim. In addition, these protocols may suffer in some scenarios because periodic protocols may not be able to cope with high rates of mobility in an ad hoc network. 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.

Zhou and Haas [65], Zapata [63], and Sanzgiri et al [56] propose the use of asymmetric cryptography to secure on-demand ad hoc network routing protocols. However, as above, 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.

Cheung [10], Hauser et al [17], and Zhang [64] describe *symmetric*-key approaches to the authentication of link-state updates, but they 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 the incoming network interface is the expected interface and that the IP TTL of the packet is 255. In a wireless ad hoc 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. Cheung [10] uses cryptographic mechanisms similar to those used in Ariadne with TESLA, but optimistically integrates routing data before it is authenticated, adversely affecting security.

A number of other researchers have also proposed the use of symmetric schemes for authenticating routing control packets. Heffernan [18] 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 [49] use symmetric primitives to secure routing between nodes and a trusted base station. Basagni et al [2] 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 [43] present work that secures against non-colluding adversaries, and they do not authenticate intermediate nodes that forward ROUTE REQUESTs, and thus do not handle authorization. Yi et al [62] discuss authorization issues. Our previous work, SEAD [21], uses hash chains to authenticate routing updates sent by a distance-vector protocol; however, that approach builds on a periodic protocol, and such protocols tend to have higher overhead than on-demand protocols and may not be suitable in highly mobile networks.

Routing protocol intrusion detection has also been studied as a mechanism for detecting misbehaving routers [7, 11, 39].

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 efficient *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, however, we carefully re-designed each protocol message and its processing. The security mechanisms we designed are highly efficient and general, so that they should be applicable to securing a wide variety of routing protocols.

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 unoptimized 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 for unoptimized DSR, and about the same on all other metrics, even though Ariadne must bear the added costs for security not present in unoptimized DSR.

We found that source-routing facilitates securing ad hoc network routing protocols. Source routing empowers the sender to circumvent potentially malicious nodes, and enables the sender to authenticate every node in a ROUTE REPLY. Such fine-grained path control is absent in most distance-vector routing protocols, which makes such protocols more challenging to fully secure.

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] Stefano Basagni, Kris Herrin, Emilia Rosti, and Danilo Bruschi. Secure Pebblenets. In *Proceedings of the Second Symposium on Mobile Ad Hoc Networking and Computing* (MobiHoc 2001), pages 156–163, October 2001.

[3] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Advances in Cryptology - Crypto '96*, edited by Neal Koblitz, pages 1–15. Springer-Verlag, 1996. Lecture Notes in Computer Science Volume 1109.

[4] 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.

[5] 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.

[6] 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, August 1994.

[7] Kirk A. Bradley, Steven Cheung, Nick Puketza, Biswanath Mukherjee, and Ronald A. Olsson. Detecting Disruptive Routers: A Distributed Network Monitoring Approach. pages 115–124, May 1998.

[8] 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 ACM/IEEE International Conference on Mobile Computing and Networking* (MobiCom'98), pages 85–97, October 1998.

[9] Michael Brown, Donny Cheung, Darrel Hankerson, Julio Lopez Hernandez, Michael Kirkup, and Alfred Menezes. PGP in Constrained Wireless Devices. In *9th USENIX Security Symposium*, pages 247–261, August 2000.

[10] Steven Cheung. An Efficient Message Authentication Scheme for Link State Routing. In *13th Annual Computer Security Applications Conference*, pages 90–98, 1997.

[11] Steven Cheung and Karl Levitt. Protecting Routing Infrastructures from Denial of Service Using Cooperative Intrusion Detection. In *The 1997 New Security Paradigms Workshop*, pages 94–106, September 1998.

[12] Tom Clark. Tom Clark's Totally Accurate Clock FTP Site. Greenbelt, Maryland. Available at ftp://aleph.gsfc.nasa.gov/GPS/totally.accurate.clock/.

[13] Don Coppersmith and Markus Jakobsson. Almost Optimal Hash Sequence Traversal. In *Proceedings of the Fourth Conference on Financial Cryptography (FC '02)*, 2002.

[14] T. Dierks and C. Allen. The TLS protocol version 1.0. RFC 2246, January 1999.

[15] Eran Gabber and Avishai Wool. How to Prove Where You Are: Tracking the Location of Customer Equipment. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 142–149, November 1998.

[16] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[17] 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, February 1997.

[18] Andy Heffernan. Protection of BGP Sessions via the TCP MD5 Signature Option. RFC 2385, August 1998.

[19] 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, August 2000.

[20] Yih-Chun Hu and David B. Johnson. Implicit Source Routing in On-Demand Ad Hoc Network Routing. In *Proceedings of the Second Symposium on Mobile Ad Hoc Networking and Computing* (MobiHoc 2001), pages 1–10, October 2001.

[21] 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), pages 3–13, June 2002.

[22] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols. Technical Report TR01-384, Department of Computer Science, Rice University, June 2002.

[23] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Efficient Security Mechanisms for Routing Protocols. In *Proceedings of the Tenth Annual Network and Distributed System Security Symposium* (NDSS 2003), February 2003. To appear.

[24] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks. In *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies* (INFOCOM 2003), April 2003. To appear.

[25] Jean-Pierre Hubaux, Levente Buttyán, and Srdjan Čapkun. The Quest for Security in Mobile Ad Hoc Networks. In *Proceedings of the Second Symposium on Mobile Ad Hoc Networking and Computing* (MobiHoc 2001), pages 146–155, October 2001.

[26] 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, 1997.

[27] 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, August 1999.

[28] 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.

[29] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In *Mobile Computing*, edited by Tomasz Imielinski and Hank Korth, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[30] 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.

[31] John Jubin and Janet D. Tornow. The DARPA Packet Radio Network Protocols. *Proceedings of the IEEE*, 75(1):21–32, January 1987.

[32] Stephen Kent, Charles Lynn, Joanne Mikkelson, 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, February 2000.

[33] 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.

[34] Young-Bae Ko and Nitin Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. In *Proceedings of the Fourth ACM/IEEE International Conference on Mobile Computing and Networking* (MobiCom'98), pages 66–75, October 1998.

[35] John Kohl and B. Clifford Neuman. The Kerberos Network Authentication Service (V5). RFC 1510, September 1993.

[36] B. Kumar. Integration of Security in Network Routing Protocols. *SIGSAC Review*, 11(2):18–25, 1993.

[37] 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.

[38] 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.

[39] Sergio Marti, T.J. Giuli, Kevin Lai, and Mary Baker. Mitigating Routing Misbehaviour in Mobile Ad Hoc Networks. In *Proceedings of the Sixth Annual IEEE/ACM International Conference on Mobile Computing and Networking* (MobiCom 2000), pages 255–265, August 2000.

[40] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgment Options. RFC 2018, October 1996.

[41] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and its Applications. CRC Press, 1997.

[42] Thomas Narten, Erik Nordmark, and William Allen Simpson. Neighbor Discovery for IP Version 6 (IPv6). RFC 2461, December 1998.

[43] Panagiotis Papadimitratos and Zygmunt J. Haas. Secure Routing for Mobile Ad Hoc Networks. In *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, January 2002.

[44] 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, August 1994.

[45] 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.

[46] Radia Perlman. *Interconnections: Bridges and Routers*. 1992.

[47] 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*, pages 35–46, February 2001.

[48] 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*, pages 56–73, May 2000.

[49] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. In *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking* (MobiCom 2001), pages 189–199, July 2001.

[50] 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.

[51] 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.

[52] Theodore S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, 1996.

[53] Yakov Rekhter and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, March 1995.

[54] 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.

[55] Pankaj Rohatgi. A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication. In *6th ACM Conference on Computer and Communications Security*, November 1999.

[56] Kimaya Sanzgiri, Bridget Dahill, Brian Neil Levine, , Clay Shields, and Elizabeth Belding-Royer. A Secure Routing Protocol for Ad hoc Networks. In *Proceedings of the 10th IEEE International Conference on Network Protocols* (ICNP '02), November 2002.

[57] Bradley R. Smith and J.J. Garcia-Luna-Aceves. Securing the Border Gateway Routing Protocol. In *Global Internet'96*, pages 81–85, November 1996.

[58] 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)*, pages 85–92, February 1997.

[59] Frank Stajano and Ross Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *Security Protocols, 7th International Workshop*, volume 1796. Springer Verlag, 1999.

[60] Trimble Navigation Limited. Data Sheet and Specifications for Trimble Thunderbolt GPS Disciplined Clock. Sunnyvale, California. Available at http://www.trimble.com/thunderbolt.html.

[61] 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*, pages 878–883, October 2001.

[62] 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.

[63] Manel Guerrero Zapata and N. Asokan. Securing Ad Hoc Routing Protocols. In *Proceedings of the ACM Workshop on Wireless Security* (WiSe 2002), September 2002.

[64] Kan Zhang. Efficient Protocols for Signing Routing Messages. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS '98)*, March 1998.

[65] Lidong Zhou and Zygmunt J. Haas. Securing Ad Hoc Networks. *IEEE Network Magazine*, 13(6):24–30, November/December 1999.