Improving the Resiliency of Ariadne

Kulasekaran A. Sivakumar and Mahalingam Ramkumar
Department of Computer Science and Engineering
Mississippi State University, MS.

***Abstract*— Ariadne is a popular secure ad hoc routing protocol based on the dynamic source routing (DSR) protocol. While Ariadne was designed to be capable of *detecting* attacks by non colluding attackers, it does not include measures to identify or narrow down perpetrators responsible for such attacks. We propose some modifications to Ariadne to improve its resiliency, by affording the ability to identify and route around malicious nodes.**

I. Introduction

A distinguishing feature of mobile ad hoc networks (MANET) [1] is their reduced dependence on infrastructure. For this reason MANET devices have to perform some additional tasks, like routing packets between nodes, that are traditionally performed by the network operators in infrastructured networks. MANET routing protocols are a set of rules to be followed by every node to cooperatively discover optimal paths and route packets between nodes. The presence of nodes that do not strictly adhere to the rules can have a deleterious effect on the overall performance of the MANET. *Secure* routing protocols strive to improve the reliability of the process under the presence of non cooperative nodes.

The ability of secure routing protocols to “live with” malicious nodes is afforded by defensive measures. For most secure routing protocols in the literature the defense lies in the ability to detect inconsistencies in routing packets (and drop such packets). However more consummate defensive measures are feasible if the routing protocol is able to identify or at least narrow down the perpetrators responsible for the observed inconsistency, and includes strategies to “route around” such nodes.

A. Contributions

In this paper we propose strategies which demand very low overheads to improve the resiliency of Ariadne [2], a secure routing protocol based on the dynamic source routing (DSR) protocol. In Section II we provide an overview of DSR and Ariadne. Though in its preferred incarnation Ariadne employs the TESLA broadcast authentication scheme, it was designed to use other forms of authentication like pairwise secrets or digital signatures. We enumerate some advantages of using pairwise secrets instead of TESLA in Ariadne.

In Section III we argue that perhaps the most compelling advantage of using pairwise secrets is that it permits the use of some DSR optimizations that are not possible when TESLA is used. We then propose two modifications to Ariadne (with pairwise secrets) consisting of measures to reduce the degrees of freedom of attackers and facilitate improved defensive measures through identification of attackers and routing around such attackers. Quantitative evaluation of the improvement in resiliency due to the additional defensive measures are estimated through simulations.

II. Background

MANET routing protocols can be classified into proactive and reactive or on-demand protocols. The dynamic source routing (DSR) protocol [3] is an on-demand protocol where a node S desiring to find a path to a node T broadcasts a route request (RREQ) packet indicating the source S, destination T, a sequence number q, and a hop-limit h_c. RREQ packets are flooded. Every node relaying an RREQ inserts its ID. Thus RREQ packets consists of *immutable* fields $\mathcal{R}_q = [S, q, D, h_c]$ specified by the source and *mutable* fields (like the IDs) inserted by the intermediate nodes. When the RREQ reaches the destination it invokes a route response (RREP) packet indicating the path through which the RREQ was received. The RREP packet is relayed back to the source along the reverse path. The source and destination may discover multiple paths as the destination can receive up to one RREQ from every neighbor.

A. Secure Routing Protocols

Most secure routing protocols mandate cryptographic authentication of disseminated routing data to facilitate measures to thwart different types of attacks. Cryptographic authentication employs security associations (SA) facilitated by key distribution schemes (KDS). Cryptographic SAs can be broadly classified into one-to-one SAs which take the form of a shared secret between every pair of entities, and one-to-many SAs, used for source authentication.

1) Application, Network and Link Layer SAs: Cryptographic SAs for MANETs can also be classified into application layer, network layer and link layer SAs. Application layer SAs are used to secure data exchanged between end-points (source and destination). It is reasonable to expect end-points to employ out-of-network mechanisms for establishing SAs. Furthermore, the KDSs used for facilitating application layer SAs should *not* (ideally) be under the control of the network operators as nodes rely on the network only for *delivering* their packets.

Network layer SAs are used for authentication of intermediate nodes (which route packets between end points). Link layer SAs are used for authentication of neighboring nodes. While end-points may have a priori knowledge of their need to interact with each other, we cannot expect a node A to foresee the need to interact with an accidental neighbor B, or some node S initiating a route request to T. Network and link layer SAs should be established in an ad hoc manner. Schemes for facilitating ad hoc establishment of network and link layer SAs will need to be controlled by off-line key distribution centers (KDC).

B. Key Distribution Schemes

Some secure extensions of DSR like SRP [4] employ only application layer SAs in the form of a shared secret between the source and the destination. Ariadne [2] employs 1) a shared application layer secret between end points; 2) a network-wide shared secret at the link layer and; 3) TESLA [5], or pairwise secrets or digital signatures for authentication of intermediate nodes (network layer).

1) Network Layer KDS: When TESLA is used a node A has access to a TESLA one-way hash chain $\mathfrak{H}_A^{T_0,\Delta} = \{K_A^0, K_A^1, \ldots, K_A^{L-1}\}$ with a commitment K_L^A, where $K_i^A = h(K_{i-1}^A)$, $h()$ is a cryptographic hash function, T_0 is an absolute value of time, and Δ is a time interval (usually of the order of a few tenths of a second). The value K_A^{L-i} is privy only to A till time $T_i = T_0 + i\Delta$, after which it can be made public. For authenticating a message M which is expected reach all verifiers *before* time T_i, the source A appends a hashed message authentication code (HMAC) $M_A^i = h(M, K_i^A)$. Any verifier with access to the certified commitment K_L^A can verify source and the integrity of the message after the source A discloses the preimage (after time T_i). As the hash chain can be used by A only in the interval T_0 to $T_{L-1} = T_0 + (L-1)\Delta$, each node may be associated with multiple hash chains. The KDC distributes certified commitments of the form $\langle K_L^A, A, T_0, \Delta\rangle$ corresponding to all nodes to every node in the network.

The preference for TESLA over digital signatures is due to the substantially high overheads required for asymmetric primitives. We shall not consider Ariadne with digital signatures any further in this paper. The reason for preferring for TESLA over pairwise secrets is that distribution of $\mathbb{O}(N)$ public values (TESLA commitments of all nodes) to all nodes in the network is practical even for large network sizes N, using a Merkle hash tree [8]. However distributing $\binom{N}{2}$ pairwise secrets, or more specifically, providing $N-1$ secrets to every node, may be impractical for large N.

2) Key Predistribution: In key predistribution schemes (KPS) every node is provided with a set of secrets (by the KDC), which permits a node to compute a shared secret with every other node, without further involvement of the KDC. Non-scalable KPSs like the "basic" KPS require each node to store $N-1$ secrets. Such schemes demand $\mathbb{O}(1)$ computational overheads for computing any pairwise secret and $\mathbb{O}(N)$ storage overheads for every node. Scalable KPSs can permit unlimited network size N by tolerating susceptibility to collusions [6]. An n-secure KPS can tolerate collusions of n nodes pooling their secrets together, irrespective of the network size N. Most scalable KPSs demand $\mathbb{O}(n)$ storage and computational overheads [6].

The reason that concerns regarding the "unsuitability of KPSs" are becoming irrelevant is due to the dramatic improvements in storage capabilities of otherwise resource constrained mobile devices. Any conceivable mobile device capable of participating in multi-hop routing will be able to afford pluggable flash storage of several GBs. Given that storing one million 64-bit secrets calls for a mere 8 MB of storage, network sizes of several tens of millions are indeed practical even for the non scalable schemes (which are not susceptible to collusions).

For most scalable KPSs the $\mathbb{O}(n)$ computational overheads is the bottle-neck which prevents realization of large n. While storage for many millions of secrets is acceptable, a million block-cipher operations (for example) is far from acceptable. However some KPSs have been proposed recently [7] for which *only the storage overhead* is $\mathbb{O}(n)$. The computational overheads for such schemes are very low (few tens of block cipher operations), and *independent* of the desired collusion resistance n, or the network size N. The high levels of collusion resistance that can be achieved renders the issue of "fragility" of scalable KPSs irrelevant in practice. Thus for scenarios where unlimited network sizes are required, scalable KPSs are good choices. For example, for the scheme in [7], achieving resistance to collusions of 100,000 entities pooling their secrets together calls for about 80 MB of storage for each node.

C. Ariadne

Ariadne provides an assurance that non colluding nodes cannot convince the end-points of the validity of a path containing malicious *insertion* of nonexistent nodes or *deletion* of nodes. Deletion attacks are addressed by using a per-hop hashing strategy. Insertion attacks are addressed by mandating intermediate nodes to append a HMAC. The network-wide secret is used to encrypt / authenticate all packets transmitted by every node to keep external nodes out of the network.

1) Per-Hop Hashing and Deletion Attacks: For a RREQ from source S to a destination T, where S and T share an application layer secret $\bar{K}_{ST}$, the source computes a HMAC $\beta_S = h(\mathcal{R}_q, \bar{K}_{ST})$, where $\mathcal{R}_q$ represents the immutable portions of the RREQ. If a scheme for facilitating pairwise secrets is used instead of TESLA the source and destination can employ the network-layer secret K_{ST} instead of $\bar{K}_{ST}$.

Apart from serving as an authentication token (verifiable only by the destination) for the immutable RREQ fields, the value β_S also seeds the per-hop hashing strategy. The seed β_S is sent along with the RREQ to all neighbors of S. A neighbor A of S replaces the value β_S with $\beta_A = h(\beta_S, A)$. A neighbor B of A relaying the RREQ onwards replaces β_A with $\beta_B = h(\beta_A, B)$ (and so on, at every hop).

When the RREQ reaches the destination T indicating a path (A, B, C, D, E, F, G), along with the per-hop hash value β_G, T can compute β_S (as T has access to $\bar{K}_{ST}$), and recursively compute β_G based on the IDs in the path. If the value β_G computed by T agrees with the value sent by G, T

can establish that the immutable RREQ fields have not been modified, and that no nodes have been deleted from the path. Note that the per hop hash value β_A is privy only to neighbors of A. Thus C, for example, cannot remove B from the path as it cannot compute $\beta_C = h(\beta_A, C)$, which is required to "trick" the destination into accepting the path as valid.

2) Insertion Attacks: If pairwise secrets are used, an intermediate node A appends a HMAC M_A based on the secret K_{AT} shared by A and the destination. Thus illegal node insertions can be detected by the destination. RREP packets are raised in response to RREQ packets free of deletion and insertion attacks. RREP packets include the list of all nodes in the path and is authenticated to the source S using the secret K_{TS}.

For Ariadne with TESLA the immutable RREQ fields include a value T_i, the time before which the RREQ should reach the destination. The HMAC M_A appended by a node A employs a secret from the TESLA hash chain of A, which will be made public only after time T_i. If the RREQ reaches the destination before time T_i, and is determined to be free of node deletion attacks, the destination waits till time T_i, and relays an RREP. The RREP includes the IDs of all nodes in the path and the HMACs appended by the nodes during the forward path. During the reverse path the intermediate nodes release the value used for computing the HMAC (which was a secret during the forward path). The RREP is authenticated by the destination using the secret $\bar{K}_{TS}$ for verification by the source S. At the end of the reverse path, the source (based on the assurance provided by the destination that the HMACs were appended by the intermediate nodes *before* T_i) can verify the HMACs. No entity claiming to be X can be inserted in the path unless the entity had access to the secrets of X's TESLA chain.

D. Pairwise Secrets vs TESLA

Some of the advantages of using pairwise secrets instead of TESLA (some of which are also enumerated in [2]) are as follows. The destination can detect *both* insertion and deletion attacks. In Ariadne-TESLA the destination can detect only deletion attacks. Detection of insertion attacks had to wait until the end of the reverse path. Secondly, the source S and destination T can also readily use a secret K_{ST} for authenticating RREQ and RREP instead of relying on an out-of-network mechanism for establishing an application layer shared secret[1] $\bar{K}_{ST}$.

Thirdly, no additional in-network bandwidth overheads are required during the RREP. For Ariadne-TESLA the pre-image has to be released by every intermediate node. To verify the preimage the source should have access to certified commitments of all intermediate nodes. In large scale networks where every node cannot store all TESLA commitments of all nodes, every intermediate node will have to attach the certified commitments along with the released pre-image. If a Merkle hash chain [8] is used by the KDC to certify S commitments, every intermediate node will have to release $\log_2 S$ hashes along with the commitment.

[1]If desired, S and T can still use $\bar{K}_{ST}$ for protecting the privacy of data packets exchanged between them after a path is established.

Fourthly, issues related to the delay sensitivity of TESLA are avoided. Note that when TESLA is used the source (which in general may not know the distance to the destination) will have to choose a conservative (large) value of time T_i to ensure that the RREQ will reach the source before T_i. Unfortunately as the pre images can be released by the intermediate nodes only after time T_i, the reverse path will have to be delayed till time T_i (even if the RREQ reached the destination well before T_i).

III. Improving Ariadne With Pairwise Secrets

Most ad hoc routing protocols, in their original incarnations, included several optimizations for improving their performance. However, as such protocols implicitly assumed that all participants are trustworthy, they were not constrained by the need to *enforce* the additional (more complex) rules associated with such optimizations. For example, the original DSR protocol includes several optimizations that employ cached routes to facilitate route response by intermediate nodes. In particular, several researchers (without considering security issues) have investigated strategies to salvage broken paths [9]-[11] during the RREP. In most secure extensions of such protocols (including Ariadne), all such optimizations cannot be used. However, one of the more compelling advantage of using pairwise secrets instead of TESLA is that it is now feasible to use some optimizations (that cannot be used in Ariadne-TESLA).

A. Re-enabling Some Optimizations

The reason that optimizations involving salvaging of routes cannot be used in Ariadne-TESLA is due to the fact intermediate nodes do not share a secret with *all* nodes. Consider a scenario where an RREP instantiated in response to an RREQ indicating a path (A, B, C, D, E, F, G) fails when D is unable to unicast the RREP to C. This situation may occur if C had moved away from the path, or if C is a malicious node which simply ignores the RREP. However, even if nodes cache RREQs only for small duration, D may have in its cache other RREQs from S indicating an alternate path, say (A, J, K, L), from S.

To use this hitherto untested partial path $D \rightarrow L \rightarrow K \rightarrow J \rightarrow A \rightarrow S$ to relay the RREP around C, it is necessary for D to share a secret with S. This is obviously not possible when we rely only an *application* layer shared secret between end-points (as in Ariadne-TESLA) as D would not have a readily available shared secret (as D has no prior knowledge of its need to establish a path to S). However, for Ariadne with pairwise secrets, D can salvage the path as it shares a network layer secret K_{DS} with S.

1) Resilience to Active Attacks: Even while active attacks involving malicious insertions or deletions (or combinations thereof) can be detected by the destination, and such inconsistent RREQs dropped, the motivation for the attacker may be to simply *preempt* other RREQs. As every node forwards only one RREQ, and the inconsistencies in the RREQ can be detected only by the destination, a bad RREQ reaching a node first can prevent propagation good RREQs that reach the node later. In general, attacks that try to preempt good RREQs with invalid RREQs are termed as rushing[2] attacks [12]. The motivation for the attacker to perform rushing attacks is obviously stronger in scenarios where the attacker can carry out such attacks in a covert manner, i.e., without facing the risk of being recognized as malicious by other nodes (like the end points or even neighbors operating in the promiscuous mode).

The strategy of routing around to salvage broken paths can also be used to improve the efficacy of the routing protocol (by reducing the need for repeated RREQs) under the presence of active attackers in the path. For example, in a scenario where the destination can detect some inconsistency in the RREQ through the path (A, B, C, D, E, F, G) and is able to determine that the perpetrator responsible for the observed inconsistency is C, the destination can still send the RREP through the path and include a request to D to route around C using cached routes (if they exist). However, while the use of pairwise secrets provides the new ability to salvage broken routes, in order the *use* this ability, the protocol should provide mechanisms for *identifying* attackers.

We now illustrate that Ariadne is susceptible to several covert attacks. The modifications introduced to Ariadne are aimed at making it possible to attribute inconsistencies in order to facilitate evasive actions.

B. Modified Ariadne

The modifications introduced are two fold. The first is the substitution of the network wide group secret used for link-level authentication in Ariadne with a one-hop secret. A node A chooses a random secret K_A and conveys the secret to all its neighbors by encrypting K_A using individually shared pairwise secrets. For example a node A with neighbors B, J and S conveys the secret K_A to the nodes as[3] $K_{AB}(K_A), K_{AJ}(K_A), K_{AS}(K_A)$, respectively. Subsequently, all RREQ packets relayed by A are encrypted using the secret K_A. The second is an additional *upstream* per-hop hash value inserted by every intermediate node.

The immutable RREQ fields include $\mathcal{R}_q$ the source S, sequence number q, the destination T, and a hop count h_c. The sequence of exchanges that occur for relaying an RREQ through a path $(A, B, \cdots)$ between S and T are as follows.

$$\begin{array}{ll}
\mathcal{R}_q = & [S, q, T, h_c],\ \beta_S = h(\mathcal{R}_q, K_{ST}) \\
S \rightarrow * & K_S([\mathcal{R}_q, \beta_S]) \\
\nu_A^S = & K_{AT}(\beta_S),\ \beta_A = h(\beta_S, A) \\
M_A = & h(\mathcal{R}_q, (A, \nu_A^S), \beta_A, K_{AT}) \\
A \rightarrow * & K_A([\mathcal{R}_q, (A, \nu_A^S, M_A), \beta_A]) \\
\nu_B^A = & K_{BT}(\beta_A),\ \beta_B = h(\beta_A, B) \\
M_B = & h(\mathcal{R}_q, (A, \nu_A^S, M_A), (B, \nu_B^A)\beta_B, K_{BT}) \\
B \rightarrow * & K_B([\mathcal{R}_q, (A, \nu_A^S, M_A), (B, \nu_B^A, M_B), \beta_B])
\end{array}$$

In the original Ariadne (with pairwise secrets), a network wide secret is used instead of secrets like K_S, K_A and K_B. The values ν are the additional values introduced in modified Ariadne. The upstream per-hop hash value β_A received by B is conveyed to the destination T by B, as $\nu_B^A = K_{BT}(\beta_A)$. The reason for encrypting the value is to protect it from downstream nodes (as the security of the per-hop hashing strategy rests on the assumption that only neighbors of A have access to the value β_A).

An intermediate node B will receive as many RREQs (from S with sequence number q) as the number of its neighbors, but forward only one. In Ariadne B will need to cache only one such RREQ relayed by B for a small duration - say a few seconds, before which it is reasonable to expect a RREP from the destination corresponding to the RREQ. In the modified Ariadne B will cache all RREQs for a few seconds, which could be used for routing the RREP around nodes that do not accept the RREQ, or on the request of the destination to avoid specific nodes in the path.

1) One Hop Group Secret: The purpose of the one-hop group secret is to thwart simple covert attacks that can exploit the shared universal group secret. Note that when only the universal secret is used at the link-level, an internal node can impersonate any other internal node, for purposes of authentication by neighbors. If a malicious node C relays a RREQ claiming to be some random "node Z", downstream nodes will not be able to detect the impersonation. While the RREQ will fail when it finally reaches the destination (as T can detect that the authentication M_Z is not consistent) it can nevertheless preempt other RREQs. Furthermore, C faces no risk in doing so. The use of unique group secrets by every node can prevent such attacks.

C. Identification of Active Attackers

To see the need for the additional value ν we shall see why some covert active attacks are still possible *without* the value ν.

1) Scenario Sans ν: Consider a scenario where C receives a RREQ (from S to T) along a path (A, B), and relays the RREQ indicating a path (Q, R, C) instead, where Q and R are fictitious nodes inserted by C. Nodes downstream of C have no reason to suspect that Q and R do not exist. Note that if C had indicated the path as (A, R, C), the upstream node B (operating in the promiscuous mode) can detect C's malicious intent. As B has access to the per-hop hash value $\beta_A = h(\beta_S, A)$, it can determine that the per-hop hash value

[2]The specific nature of the invalidity of the rushed RREQ can take several forms like active attacks involving insertions and / or deletions, forwarding RREQs received through one-way links, etc.

[3]In the rest of this paper $C = K(M)$ denotes symmetric encryption of a message M using a secret K, and $M = K^{-1}(C)$ denotes decryption.

reported by C is inconsistent with the path indicated (as C does not have access to β_S or β_A). However, in this scenario, B has no reason to suspect C, as B does not have access to β_S and hence $\beta_Q = h(\beta_S, Q)$ or $\beta_R = h(\beta_q, R)$. Furthermore (from B's perspective) it is entirely possible that C *does* have a neighbor R.

Assume that the destination receives the request indicating a path (Q, R, C, D, E, F, G). The destination T can detect that the per-hop hash value β_G submitted by G is inconsistent with β_S and the path. However the destination learns little else. All that T can conclude with certainty is that node G exists (as T can verify verifying M_G). As T does not have access to the value β_F used by F to compute the HMAC M_F, T cannot verify the HMAC M_F, and thus cannot even determine if the node F actually exists in the path.

If T desires to determine *who is responsible* for perpetrating this attack, it can come to several likely conclusions: a) G is a malicious node and every other node in the path has been maliciously inserted by G; b) G is a good node, but F may[4] have maliciously inserted nodes (Q, R, C, D, E) in the path; c) Both G and F are good nodes and the node E may have inserted nodes (Q, R, C, D) in the path, and so on. In other words, all that the destination learns is that "some node in the path is malicious." Furthermore there is no viable strategy to rectify the situation. While the source may invoke a second RREQ after the first RREQ times out, the end result may be the same (barring topology changes).

2) Scenario With ν: Now consider the scenario where every node appends the upstream per-hop hash value. The three values appended by every node (for example, the values (C, M_C, ν_C^R) appended by C) permits the destination T to verify the *self-consistency* of the values. The three values indicated by C (if self-consistent) convey to the destination that a node C (which has access to the secret K_{CT}) actually exists in the path, and claims that a value $\beta_R = K_{CT}^{-1}(\nu_C^R)$ was sent by its "upstream node R." However, such a claim is not verifiable by T unless the HMAC M_R appended by R is consistent with β_R. If this is not the case, there are two possible explanations for the inconsistency

1) A malicious C could have chosen a random β_R and appended values (M_C, ν_C^R) consistent with the random β_R.
2) A malicious R could have advertised random β_R (as C claims).

In either case, the upstream per-hop hash value enables the destination to eliminate the nodes between the last self-consistent node and the destination as possible perpetrators.

D. Corrective Measures to Improve Resilience

When any inconsistency is detected by the destination, it proceeds to verify the self-consistency of the values appended by every node in the path, starting with the node closest to the destination. In this scenario where C was an active attacker, depending on the C's strategy, the destination can conclude that nodes D, E, F, G or nodes E, F, G are "beyond blame." If C had chosen a random β_R and appended consistent (M_C, ν_C^R), then C is the last consistent node, and it is desirable to avoid C and R during the RREP. The destination can instruct D to deliver the RREP to S using a cached path that does not include R or C. Alternately, if C had appended non self-consistent (M_C, ν_C^R) (and D is the last self-consistent node), the destination can request E to route the RREP around C and D.

The RREP, along with the warning (say) that R and C should be avoided, is authenticated individually to the source and all intermediate nodes between C and the destination T (nodes D, E, F, G). Now D uses its other cached RREQs from S to determine an alternate path to S that does not include R or C. This "corrective measure" can improve the chances of discovering an alternate path free of malicious nodes, without the need for a second RREQ.

In scenarios where no alternate path can be discovered by D, a second RREQ is raised by S (after the first RREQ times out). During this time however, nodes like D, E, F, G (downstream of C and R) will simply drop RREQs from S to T that include C or R in the path. This improves the chances of other RREQs (which were preempted the first time) reaching the destination. A warning from the destination indicating that "R and C should be avoided," is *not* construed by other nodes that R or C is necessarily malicious. Nodes which receive this warning will simply interpret it as a request from T to avoid C and R for RREQs in which T is the destination. T has every right to make this request even without offering conclusive proof of misbehavior by R or C. Thus nodes like D, E, F, G will *not* drop RREQs relayed by C or R which indicate destination other than T.

1) Evaluation of Improvements in Resilience: Simulations were carried out to evaluate the practical utility of strategies for narrowing down possible perpetrators and discovering alternate paths by estimating the fraction of RREQs that succeed under the presence of active attackers, with and without the use of the additional upstream per-hop hash value.

Sets of 200 nodes were simulated (with uniformly distributed x and y coordinates) in a square region with unit edges. The range of the nodes was chosen as 0.1 units (each node had 5 neighbors on an average). Of the two hundred nodes, b randomly chosen nodes were labeled malicious. RREQ propagation was simulated between every pair of nodes. On the whole, three different realizations of the network were simulated with different sets and numbers of bad nodes. The simulation results are depicted as fraction of nodes that succeed in discovering a path free of bad nodes (y-axis) vs the shortest number of hops between the source and destination (between which RREQ propagation was simulated). RREQ propagation was simulated for over 400,000 pairs separated by hop lengths between 4 and 10.

The quest to discover a path between two nodes is assumed to succeed if at least one of the paths established is free of malicious nodes. In Figure 1 plots labelled S1 correspond

[4]If G is a good node F should exist as G would have verified the link-level authentication appended by F.

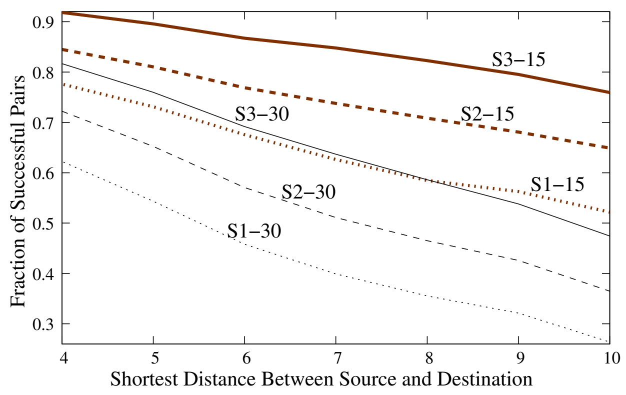

Fig. 1. Quantitative measures for the utility of the upstream per-hop hash value.

to the scenario where the additional upstream hash is not used. The plot indicates the fraction of node pairs which can establish a path free of any of the b bad nodes. Simulation results are reported for $b = 15$ (S1-15) and $b = 30$ (S1-30).

In the scenario where the upstream hash is used, in our simulations we assumed that in a path with one or more bad nodes, destination will "identify" the bad node closest to the destination, and its immediate neighbor (downstream or upstream, chosen randomly) as nodes to be "avoided." The node downstream of the pair of nodes specified by the destination will succeed in salvaging a path if 1) an alternate path (based on other RREQs received) exist that do not include the nodes specified by the destination; and 2) the alternate path is free of any of the b bad nodes.

The plots labelled S2 indicate the fraction of node pairs which succeed in finding a path without requiring a second RREQ from the source. In scenarios where even the attempt to slavage RREP fails, the source invokes a second RREQ. The plots labelled S3 indicate fraction of successful node pairs (either the first or the second RREQ attempt succeeds).

As can be seen from the simulation results the success rate after the second RREQ for the scenario with 30 bad nodes (S3-30) is comparable to the success of the first RREQ with just 15 bad nodes (S1-15). It is important to note that in the absence of warnings from the destination (which depends on the ability of the destination to narrow down perpetrators - provided by the additional value ν) the second RREQ has only as much chance of succeeding as the first. Thus, in this particular instance, it could be argued that the modifications proposed provide Ariadne with the ability to resist twice as many malicious nodes.

IV. Conclusions

We proposed some improvements to Ariadne with pairwise secrets to improve its ability to tolerate malicious nodes. We argued why the use of a scheme for pairwise secrets is indeed practical, and enumerated several compelling advantages of using pairwise secrets instead of TESLA in Ariadne. In our opinion, the most compelling advantage is that a scheme for establishing private channels between nodes permits more comprehensive defensive measures. The specific modifications suggested included 1) using such private channels to establish a private neighborhood with the intent of overcoming several simple covert attacks and providing the ability for nodes to cut off neighbors suspected of malicious intents; 2) conveying the upstream per hop hash value securely (while hiding it from downstream nodes) to the destination to aid the destination in narrowing down possible perpetrators.

References

[1] Web Link, http://www.ietf.org/html.charters/manet-charter.html.

[2] Y-C Hu, A Perrig, D B.Johnson, "Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks," Journal of Wireless Networks, **11** pp 11–28, 2005.

[3] P. Johanson, D. Maltz, "Dynamic source routing in ad hoc wireless networks," Mobile Computing, Kluwer Publishing Company, 1996, ch. 5, pp. 153-181.

[4] P Papadimitratos, Z. J.Haas, "Secure Routing for Mobile Ad Hoc Networks," Proceedings of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference(CNDS 2002), San Antonio, Texas,2002.

[5] A. Perrig, R. Canetti, D. Song, D. Tygar, "Efficient and Secure Source Authentication for Multicast," in Network and Distributed System Security Symposium, NDSS '01, Feb. 2001.

[6] R. Blom, "An Optimal Class of Symmetric Key Generation Systems," *Advances in Cryptology: Proc. of Eurocrypt 84*, Lecture Notes in Computer Science, **209**, Springer-Verlag, Berlin, pp. 335-338, 1984.

[7] M. Ramkumar, "Trustworthy Computing Under Resource Constraints With the DOWN Policy," IEEE Transactions on Dependable and Secure Computing, **5** (1), pp 49–61, Jan-Mar 2008.

[8] R.C. Merkle "Protocols for Public Key Cryptosystems," IEEE Symposium on Security and Privacy, 1980.

[9] R. Bai, M.Singhal, "Salvaging route reply for on-demand routing protocols in mobile ad-hoc networks," Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems, Montreal, Quebec, Canada, 2005.

[10] R.Duggirala, R.Gupta, Q.A.Zeng, D.P.Agarwal, "Performance Enhancements of AdHoc Networks with Localized Route repair," IEEE Transactions on Computers, Vol 52, no.7, pp. 854-864,2003.

[11] J.S.Youn, J.Lee, D.h.Sung, C.H.Kang, "Quick Local repair Scheme using Adaptive Promiscuous Mode in Mobile Ad Hoc Networks,"**1**(1), pp 1-11, May 2006

[12] Y-C Hu, A. Perrig, D.B. Johnson, "Rushing Attacks in Wireless Ad Hoc Network Routing Protocols," WiSe 2003, San Diego, CA, September 2003.