Defending against Packet Injection Attacks in Unreliable Ad Hoc Networks

Qijun Gu*, Peng Liu†, Sencun Zhu†, and Chao-Hsien Chu†
* Department of Computer Science, Texas State University, San Marcos, TX 78666
† School of Information Sciences and Technology, Pennsylvania State University, University Park, PA 16802

Abstract— **Ad hoc networks are usually unreliable and have limited bandwidth resources. In such networks, packet injection attacks can cause serious denial-of-service via wireless channel contention and network congestion. To defend against this type of injection attacks, we propose SAF, an efficient and effective Source Authentication Forwarding protocol. The protocol can either immediately filter out injected junk packets with very high probability or expose the true identity of an injector. Differing from other forwarding defenses, this protocol is designed to fit in the unreliable environment of ad hoc networks. Our simulation shows that SAF incurs very lightweight overhead in communication and computation.**

I. Introduction

Ad hoc networks are usually unreliable and have limited bandwidth resources. In such networks, attackers can bring serious denial-of-service via congestion by injecting junk packets. Compared with other types of DoS attacks in ad hoc networks, packet injection attacks in general are easier for an attacker to launch but are more difficult for us to defend against, because an attacker may claim to be a forwarding node instead of a source node. To prevent this type of attacks, a forwarding node needs to filter out the injected junk packets as early as possible, not leaving it for the destination to detect. The longer time a junk packet stays in the network, the more congestion it can cause.

Due to the lack of source authentication during data packet forwarding, in many ad hoc network communication protocols, an attacker can inject junk packets into a route, even if the route is established by secure routing protocols [1], [2], [3]. Hence, source authentication is needed so that every node is able to verify the authenticity of the data packets it is forwarding. Since digital signing every packet is too expensive for ad hoc networks, this paper presents SAF, a symmetric key-based hop-by-hop source authentication forwarding protocol. In SAF, a source node secretly sets up a pairwise key with each en route node. When the source sends a data packet, it computes an authentication token for each en route node using the pairwise key. An en route node forwards a data packet only if it is authenticated. Thus, only data packets from the real source can go through the route and reach the destination. This approach, although simple, can provide immediate source authentication and thus inherently supports the on-demand nature of ad hoc networks. For the successful deployment of SAF, however, we must address several practical challenges such as route change and packet disorder caused by the unreliability nature of an ad hoc network.

Contributions First, the junk packet injection attack studied in this paper differs from false data packet injection or packet dropping attacks in the literature. In this attack, attackers wants to inject junk packets to cause congestion instead of providing false information. Hence, the main defense goal is to ensure that a route only serves the legitimate packets by preventing attackers from creating extra junk packets or replaying legitimate packets in the route. Second, the proposed protocol is specially designed to handle various problems in the forwarding procedure in an unreliable ad hoc network. We propose a new authentication scheme to let en route nodes take the responsibility in authentication when a route is broken. As we show later, SAF not only provides the defense against packet injection attacks, but also ensures the normal delivery of legitimate data packets. Third, we systematically analyze and summarize various problems when applying source authentication in forwarding data packets in ad hoc networks. Misuse of the proposed protocol is against attack objectives, and does not affect non-misused packets.

The rest of the paper is organized as follows. Section II describes the related work. Section III presents the design of SAF protocol in unreliable ad hoc networks. In Section IV, we analyze its security against injection attacks and whether misuse of this protocol will cause other problems. We evaluate SAF in Section V. Finally, we conclude in Section VI.

II. Related Works

Many previous work on ad hoc network security focused on secure routing[1], [3]. As a contrast, SAF focuses on filtering junk packets injected into the routes established by these secure routing protocols. Two most related approaches [4], [5] are proposed for filtering false data packets in sensor networks. Due to the difference between ad hoc and sensor networks and the different defense objectives, these approaches cannot be simply applied in our study. In [4], Ye *et al.* proposed a statistic filtering scheme that allows en route nodes to filter out false data packets with some probability. However, this approach cannot prevent attackers from replaying packets. It is also possible that junk packets can go through the network (although they will be discarded at the destination) if they do not carry keys that the en route nodes have. Zhu *et al.* [5] proposed an interleaved hop-by-hop authentication scheme that guarantees to detect and drop false data packets within a certain number of hops. Nevertheless, due to unreliability in mobile ad hoc networks, the interleave association cannot sustain in routes that dynamically change. Hence, this scheme cannot be applied in our study either. Due to space limit, we cannot enumerate all other related defense approaches in the literature. However, we notice that many solutions are not suitable against such attacks, since routing nodes in ad hoc networks are not trustable, or junk packets cannot be immediately discarded, etc. We intend to prevent and limit the attack impacts, instead of letting the junk packets travel through multiple hops to affect other areas of an ad hoc network.

III. Protocol Design

In this section, we present the SAF protocol. First, we summarize the assumptions. Then, we present the basic hop-by-hop source authentication. We show that unreliability makes source authentication hard in ad hoc networks. Finally, we present the complete scheme of SAF.

A. Network Assumptions and Attack Models

This paper mainly studies unicast communication. SAF is designed to work with the routing protocol DSR [6], since it needs IDs (i.e. the node's address) of en route nodes along the forwarding path. Other protocols, such as AODV [7], can be applied with our protocol with extensions to carry IDs of en route nodes in the routing packets. In the paper, we also consider a complex environment in ad hoc networks. For example, a packet could be lost due to transmission error, a route could be broken due to power down of a routing node, etc. SAF is designed to fit in such an unpredictable and unfriendly environment. In an injection attack, an attacker may be a compromised node, or simply a malicious node. Because it is risky for an attacker to misbehave its own ID, the attacker will impersonate other nodes. An attacker may eavesdrop on all traffic, replay older packets, or modify overheard packets and re-inject them into the network. Furthermore, multiple attackers may collude in attack.

B. Hop-by-Hop Source Authentication

As we discussed, to prevent packet injection attacks, hop-by-hop source authentication is a good choice, in which three steps are needed. First, the source node sets up pairwise keys with its en route nodes. Second, the source node authenticates its packets for each en route node with the pairwise keys. Finally, each en route node verifies packets upon receiving them. If packets are authentic, en route nodes forward them; otherwise, discard.

1) Pairwise Keys Establishment: The source node sets up a pairwise key with every en route node along the path, based on IDs of routing nodes obtained from DSR route reply packets. The simplest way is to pre-load pairwise keys into nodes, although the memory requirement is not tolerable when the network is large. The literature provides many novel key management schemes with better performances. For example, in random key schemes [8], [9], any two nodes can establish a pairwise key with a sufficiently high probability. Hop-by-hop source authentication can be based on any of these schemes as long as they can ensure the secure pairwise key establishment between nodes over multiple hops.

2) Authentication: For discussion, we denote that a source node S sends data packets[1] to a destination node D through a route of $n-1$ routing nodes, which are ordered as $R_1, ..., R_j, ..., R_{n-1}$, and R_n is D.

When S wants to send data packets to D, S attaches an authentication header $A(i)$ to each data packet $PKT(i)$.

$$A(i) = SID(i)||PC(i)||\delta_{R_1}(i)||\delta_{R_2}(i)||...||\delta_{R_n}(i)$$

Where $SID(i)$ is the source ID, $PC(i)$ is packet count, and $\delta_{R_j}(i)$ is the authentication token for en route node R_j.

Each token $\delta_{R_j}(i)$[2] is computed based on a keyed-hash as $H_{k_{SID,R_j}}(PC(i), L(i,j))$. Here, k_{SID,R_j} is the pairwise key shared only between SID and R_j. $L(i,j)$ is the packet size including the data and the remaining authentication header at R_j. $H(*)$ is a secure hash function. The token design ensures that each en route node R_j can only verify one token, and the packet count $PC(i)$ and the packet size $L(i,j)$ are secured in the tokens.

3) Forwarding: In a route, each node sets a forwarding entry, which extends a routing entry to store information for verifying received packets. The entry records the packet count PC_{last} in the last received packet and the initial packet count obtained from the first received packet. The second count is used to handle unreliability as discussed later.

Upon receiving $PKT(i)$, R_j must verify $A(i)$. First, R_j checks whether the count $PC(i)$ is greater than PC_{last}. If yes, then R_j recomputes its token $\delta_{R_j}(i)$ based on the claimed packet source, packet count and packet size[3]. If the computation result matches the token carried in the packet, R_j removes tokens (if any) of current and previous hops from $A(i)$, records $PC(i)$ as PC_{last}, and forwards the data packet to the next hop R_{j+1}. Otherwise, if either of the previous two steps fails (i.e. the token is wrong or $PC(i) \leq PC_{last}$), R_j discards the packet.

In brief, such a forwarding procedure can achieve the following defense goals.

- A source must use its own ID to authenticate a packet in order that the packet can be delivered in its route.
- An authenticated packet cannot be replayed in the same route.
- An authenticated packet cannot be played in any other route which is not set by the source.
- Attackers cannot insert junk bits into packets for congestion purpose.

Note that the above design does not provide data integrity because the computation is only over the packet count and the

[1]Data packets include all packets with IP headers, but exclude the routing packets and the keying packets. The later two types of packets are for route discovery and pairwise key setup, and secured by their own protocols.

[2]The length of a token is determined by the trade off between security and performance. For discussion, a token has 8 bits in this paper, although the hash output could be 256 bits or longer.

[3]Because each en route node knows how many tokens should be left when it receives a packet, R_j calculates the packet size by adding the data size and the size of the remaining authentication header that should be at its own hop, in stead of directly obtaining the packet size from the received packet. This approach is applied to address verification in unreliability.

packet size. However, this does not conflict with our design goal, because the modification of packet content does not cause congestion in the network. Moreover, this design is very computationally efficient because the keyed-hash is always computed over a very short piece of information. Nevertheless, this scheme can be easily modified to provide data integrity as well if desired.

C. Unreliability

In the following, we discuss how unreliability (route change and packet disorder) may fail hop-by-hop source authentication.

1) Route Change: In an ad hoc network, a new route segment may be set up due to various reasons. For example, in Figure 1, the new route (dashed line) diverges from the previous one (solid lines) at node 2 due to the link failure between nodes 2 and 3. Because S may not know the new route immediately when the old route is broken, nodes 6 and 7 will not have any pairwise key with S. In addition, some data packets may be already buffered in nodes 1 and 2 to be forwarded, and S cannot modify the authentication headers in these packets. Hence, these buffered packets may be discarded even after S sets a new route. Note that nodes 3, 4 and 5 can still use the old segment to forward their buffered packets, since the old segment is still valid at their positions and their buffered packets have valid authentication headers.

2) Packet Disorder: Packet count PC in fact represents the order of the packet delivery. If the order is mixed or reversed, a packet with a smaller PC will be discarded. We notice that the disorder can be caused by two reasons: either an attacker intentionally changes the order of the packets, or a route is changed. We will discuss the first reason in Section IV. The second reason can also be illustrated in Figure 1, where the new and the old routes overlap after node 4. Assume that node 3 is congested for a long time after the new segment is discovered. Hence, the packets going through node 7 will reach node 4 before the old packets buffered in node 3 do. Because the packets buffered in node 3 have smaller PCs, they will be discarded by node 4 if node 4 only records the PC_{last} from the packets forwarded by node 7 as in the basic hop-by-hop source authentication.

D. Source Authentication Forwarding

The idea to solve the problems caused by unreliability is to let the first en route node in the new route segment take the responsibility of the real source to start another forwarding procedure. As shown in Figure 1, node 2 becomes the "source" of the new route segment, and appends its own authentication header A^2 to packets, which is similarly computed as A^S, and SID in A^2 is node 2's ID. We name such an en route node as a **starter**, and SID in the new authentication header is thus the starter ID instead of the source ID.

In SAF, an en route node maintains a forwarding entry for each new route segment, in which the node records the corresponding initial packet count obtained from the first received packet in the segment. Thus, the node has multiple packet count intervals. The node compares the PC of a received packet in the corresponding interval. For example, in Figure 1, node 4 obtains two initial PCs from A^S and A^2, denoted as PC^S and PC^2. Obviously, $PC(i')$ in any packet buffered in node 3 satisfies $PC^S \le PC(i') < PC^2$; while $PC(i)$ in any packet going through node 6 satisfies $PC^2 \le PC(i)$. Hence, node 4 actually knows two intervals: $[PC^S, PC^2)$ and $[PC^2, \infty)$. Node 4 also knows the last packet count for each interval, denoted as PC^S_{last} for $[PC^S, PC^2)$ and PC^2_{last} for $[PC^2, \infty)$. When node 4 receives a packet $PKT(i)$, it compares $PC(i)$ in the packet with either PC^S_{last} (if $PC(i)$ falls in $[PC^S, PC^2)$) or PC^2_{last} (if $PC(i)$ falls in $[PC^2, \infty)$).

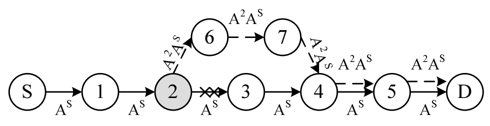


Fig. 1. Starter in a new route

The node verifies the tokens (if any) for corresponding route segments as discussed before. However, the packet size used in authentication is changed to the sum of the data size, the size of the remaining authentication header of the segment, and the number of previous authentication headers. The last parameter in the packet size shows how many headers already exist when the current header was computed by its starter, and it is used to ensure that no extra junk header can be easily inserted into packets. For example, in Figure 1, if node 4 receives a packet from node 7, it can verify both A^2 and A^S in the packet. A^2 must be the last authentication header, and no other header should appear after A^2. Also, when A^2 was computed by node 2, A^S already exists. If A^S is removed or a junk header is inserted, the verification of A^2 in node 4 will fail. Finally, because the packet going through node 7 is not verified by node 3, the token for node 3 in A^2 is not removed by node 3. Since node 4 can verify A^2, it will remove all previous tokens from A^2 before it forwards the packet as discussed before.

Note that node 1 may not have any information about either the new segment or the forwarding procedure in the new segment. Node 1 may work as if nothing happened in the route. This new forwarding procedure works until S knows the new route and resets forwarding[4].

IV. Security Analysis

For congestion, an attacker may inject junk packets into routes or insert junk bits into legitimate packets. To inject an extra junk packet, the attacker needs to provide a valid token for a larger packet count. As we notice that, a token is only a few bits of the hash output. Nevertheless, the attacker does not know the pairwise key that is only shared between the starter and the corresponding en route node. To break the token without knowing the pairwise key is as difficult as to break the hash function. Hence, if the attacker wants to inject a junk packet, the only way is to guess the token.

[4]For example, in DSR, node 2 sends a "gratuitous" Route Reply to S that contains the IDs of the routing nodes in the new route. Hence, S can start a new forwarding procedure in the new route.

Since the successful probability is very small (depending on the length of the token), SAF effectively filter out junk packets. Furthermore, if an attacker frequently sends guessed tokens in a route, he easily exposes himself, since other en route nodes can easily detect frequent verification failures.

Since the data size, the authentication header size and the number of previous authentication headers are protected in tokens, an attacker, if being an en route node, cannot insert junk bits into packets. However, an attacker, if claiming to be a starter, can put junk bits in packets by forging some authentication headers for non-existing route segments. Such an injection is in fact "legal" in the route, since the attacker does use its own identity to authenticate the packets. This situation also exists in legitimate traffic. However, at least the destination node can detect such an injection, because the destination node knows all valid routes. Hence, the attacker exposes himself as well.

In the following, we present the major properties of SAF that show how SAF prevents injection and the consequences of misuse of SAF. In summary, it is not easy for an attacker to inject junk packets, unless the attacker uses its own ID in attack. Also, misuse of SAF by dropping, replaying, disordering or modifying authentication headers in general results in the drop of misused data packet, but does not affect other legitimate data packets. Hence, misuse is against the objective of packet injection attacks in terms of congestion. The detailed analysis on the properties are not presented due to the limit of space.

Property 1: If an attacker claims to be a starter (or a source), it must use its own ID to authenticate injected junk packets.

Property 2: If an attacker is an en route node, the probability that a forged packet can survive is negligible. Specifically, the probability that a forged packet can go through m hops is only $(\frac{1}{256})^m$, if a token has 8 bits.

Property 3: If an attacker intentionally modify the authentication header, the result is the same as that the attacker drops the packet.

Property 4: If an attacker replays authentication headers in any junk packet, the packet will be discarded.

Property 5: If an attacker disorders the packets to be forwarded, the result is the same as that the attacker simply discards these disordered packets.

V. Performance Evaluation

In this section, we use simulations in NS2 [10] to evaluate the performance of SAF. We examine how much overhead this protocol brings to each routing node in an ad hoc network. In the following, we first present the detail settings for the simulations, and then illustrate and discuss the impact of this protocol on data forwarding.

A. Simulation settings

1) Network and traffic: The widely deployed IEEE 802.11 is used as the MAC and PHY protocols. By default in NS2, the channel capacity is $1Mbps$, and each node has a transmission

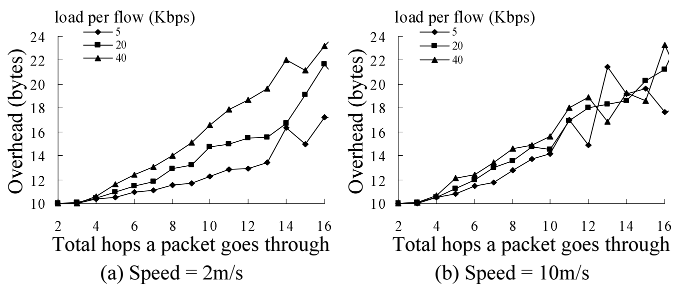


Fig. 2. Communication overhead per hop

range of 250 meters. For communications over multiple hops, DSR is used as the routing protocol. We use the scenario generation tool in NS2 to generate various network topologies in a $1500m \times 1500m$ area. In each simulation, 100 nodes and 10 connections are randomly put in the network. Nodes move randomly at the maximum speed of $2m/s$ or $10m/s$. Each connection picks a random time during the first 5 seconds to start its traffic, and all traffic lasts 60 seconds. The load of each connection is $5Kbps$, $10Kbps$, $20Kbps$, $30Kbps$ or $40Kbps$, and the payload of a data packet is 512 bytes. In each scenario, the loads of all connections and the maximum speeds of all nodes are the same. In each scenario, we randomly generate 10 cases to get average measurement.

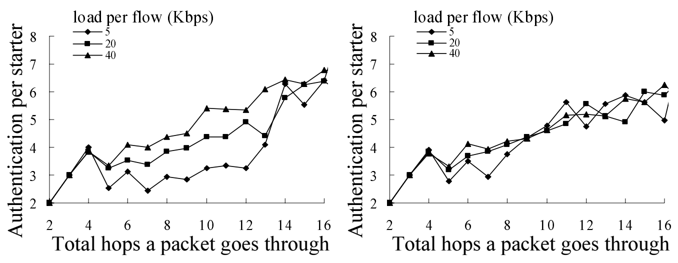
2) Performance Metrics: We measure four performance metrics of SAF in all scenarios. (1) *Communication overhead per hop* is measured as the number of bytes that are carried to each data packet. (2) *Authentication per starter* is measured as the number of authentication tokens that a starter computes to authenticate a data packet. (3) *Verification per hop* is measured as the number of authentication tokens that an en route needs to verify a data packet. (4) *Data throughput per flow* is measured as the data rate (Kbps) of the data forwarding with SAF.

B. Impacts of SAF on Regular Traffic

As discussed in Section II, existing approaches cannot be applied in the situation we are considering, because they do not provide desired security or robustness. Hence, SAF is mainly evaluated in terms of its overheads in packet forwarding. Its effectiveness in terms of defense is illustrated by its properties. We present major observations from the simulations in the following.

1) Overhead of SAF: SAF appends an authentication header of several bytes to every data packet, which include SID (4 bytes), PC (4 bytes), and authentication tokens (1 byte/token). The size of an authentication header changes along the path[5], as en route nodes remove corresponding tokens from a packet when they forward a packet, or starters add new authentication headers to a packet for a new route segment. As shown in Figure 2, more unreliable the network is, more segments a packet goes through and thus more headers are appended. When the load is light (5Kbps) and the speed

[5]A path means all hops a data packet goes through until reaching the destination. Hence, a path may contain several segments, each of which is set up when a new route is used to replace the broken one.



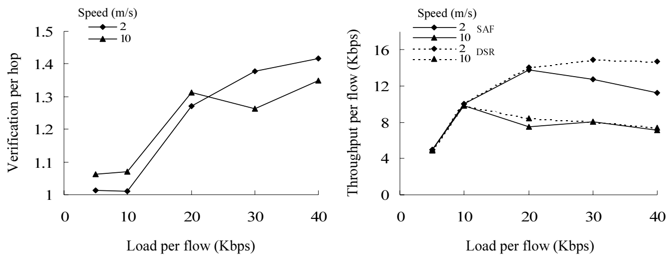
(a) Speed = 2m/s (b) Speed = 10m/s

Fig. 3. Number of authentication tokens a starter needs to compute

is low (2m/s), a path with one more hop only adds 0.5 bytes to the average overhead. On the other hand, when the load and the speed are high (40Kbps and 10m/s), a path with one more hop could add more than 1 byte to the average overhead. Especially, when speed is high, unreliability is less affected by load, because route change is more frequently. Nevertheless, compared with the sizes of payload, IP header and MAC header, the overall overhead of SAF is lightweight around 10 (in 2-hop paths) to 24 bytes (in 16-hop paths) in our simulation.

2) Computation of SAF: The starter needs to compute authentication headers for data packets and each en route node needs to verify sources. As depicted in Figure 3, unreliability (higher load and speed) also increases the computation for starters (although slightly). In the worst case (40Kbps and 10m/s), a starter needs to compute around 0.3 authentication token in average for each hop in the path. The computation cost for each en route node is depicted in Figure 4(a), where load shows to be a more important factor than speed. When the load is light (5Kbps to 10Kbps), a little more than 1 verification is needed in each hop for each data packet. When the load is between 10Kbps and 20Kbps, the verification quickly increases from 1.05 to 1.3. Then the increase is slowed down as the load is more than 20Kbps. Note that the maximum verification is less than 1.5 even in the very unreliable situation. This result, combined with the overhead, indicates that even if a data packet may carry a big authentication header with many authentication tokens, each en route node may only find one or two tokens that are designated to it in an unreliable environment.

3) Throughput of SAF: Finally, we use Figure 4(b) to address the major concern on whether SAF will affect the throughput. For comparison purpose, we also conduct simulations, where only regular DSR is used. In the figure, throughput of SAF is represented by solid lines, and DSR by dashed lines. SAF does not interfere with DSR when the network is reliable, i.e. light loads and low speeds. When the load is more than $20Kbps$, the network becomes unreliable. The overhead of SAF only slightly reduce the throughput, although some overheads are added in the network. Figure 4(b) also shows that SAF is practical in an unreliable ad hoc network. The solid lines demonstrate that SAF can work even when more than 62% of packets are dropped. Hence, SAF not only protects the network, but also does not interfere with normal network traffic.



(a) Number of authentication tokens an en route node needs to verify (b) Throughput comparison

Fig. 4. Verification Cost and Throughput

VI. Conclusion

To defend against packet injection DoS attacks in ad hoc networks, we present SAF, a hop-by-hop source authentication protocol in forwarding data packets. The protocol can either immediately filter out injected junk packets with very high probability or expose the true identity of the injector. This protocol is designed to fit in the unreliable environment of ad hoc networks. For each data packet, the protocol adds a header of a few bytes for source authentication. Every en route node needs to verify less than 1.5 authentication tokens for each packet even when the network is very unreliable. Hence, the protocol is lightweight and almost does not interfere with regular packet forwarding. As future works, the communication overhead of SAF can be further reduced by applying techniques such as bloom filter.

Acknowledgement

This work was supported in part by the ARO grant W911NF-05-1-0270.

References

[1] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: a secure on-demand routing protocol for ad hoc networks," in *ACM MobiCom*, 2002, pp. 12–23.

[2] M. G. Zapata and N. Asokan, "Securing ad hoc routing protocols," in *ACM workshop on Wireless Security*. Atlanta, GA, USA: ACM Press New York, NY, USA, 2002, pp. 1–10.

[3] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Sead: secure efficient distance vector routing for mobile wireless ad hoc networks," *Ad Hoc Networks*, vol. 1, no. 1, pp. 175–192, 2003.

[4] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical en-route detection and filtering of injected false data in sensor networks," in *IEEE Infocom*, 2004.

[5] S. Zhu, S. Setia, S. Jajodia, and P. Ning, "An interleaved hop-by-hop authentication scheme for filtering false data in sensor networks," in *IEEE Symposium on Security and Privacy*, Oakland, California, 2004.

[6] D. Johnson, D. Maltz, Y. C. Hu, and J. Jetcheva, "The dynamic source routing protocol for mobile ad hoc networks (dsr), ietf internet draft, draft-ietf-manet-dsr-09.txt," Feb. 2002.

[7] C. Perkins, E. Royer, and S. R. Das, "Ad hoc on-demand distance vector (aodv) routing, ietf internet draft, draft-ietf-manet-aodv-11.txt," June 2002.

[8] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks," in *ACM CCS*, 2003, pp. 42–51.

[9] S. Zhu, S. Xu, S. Setia, and S. Jajodia, "Establishing pairwise keys for secure communication in ad hoc networks: a probabilistic approach," in *IEEE ICNP*, 2003, pp. 326–335.

[10] NS2, "The network simulator, http://www.isi.edu/nsnam/ns/," 2004.