Defense Against Packet Injection in Ad Hoc Networks

Qijun Gu[1] Peng Liu[2] Chao-Hsien Chu[2] Sencun Zhu[3]

[1]Department of Computer Science
Texas State University, San Marcos, TX 78666
[2]School of Information Sciences and Technology
Pennsylvania State University, University Park, PA 16802
[3]Department of Computer Science and Engineering
Pennsylvania State University, University Park, PA 16802

Abstract

Wireless ad hoc networks have very limited network resources and are thus susceptible to attacks that focus on resource exhaustion, such as the injection of junk packets. These attacks cause serious denial-of-service via wireless channel contention and network congestion. Although ad hoc network security has been extensively studied, most previous work focuses on secure routing, but cannot prevent attackers from injecting a large number of junk data packets into a route that has been established. We propose an on-demand hop-by-hop source authentication protocol, namely SAF, to defend against this type of packet injection attacks. The protocol can either immediately filter out injected junk packets with very high probability or expose the true identity of an injector. Unlike other forwarding defenses, this protocol is designed to fit in the unreliable environment of ad hoc networks and incurs lightweight overhead in communication and computation.

Keywords: Packet injection, Source authentication, Secure forwarding, Denial of Service, Ad hoc network, Wireless Security

1 Introduction

Ad hoc networks are usually unreliable and have limited bandwidth resources. In such networks, attackers can cause 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 protocols, an attacker can inject junk packets into a route, even if the route is established by secure routing protocols [9, 33, 11]. The injected packets not only impact on the legitimate traffic in the injection route, but also on the traffic nearby the injection route, because wireless communication is open on air. The common consequence is the congestion in the areas along the injection routes. The injection attack is as effective as the other DoS attacks (including signal jamming [31], carrier sense congestion [2], packet dis-routing [9], packet dropping or reordering [1]). Our previous work [7] shows that the injection attack does not demand a large amount of attacking resources, and a small number of attackers can cause significant throughput degradation network-wide. We find that injection in multi-hop routes have a more serious impact on legitimate traffic than simply congesting neighboring nodes. In Appendix A, we provide some major findings from our previous study on injection attacks.

Due to the easiness of attack approach and the severity of attack consequence, more research is needed to defend against injection attacks. In this study, public key based source authentication is not considered, because signing and verifying every data packet is too expensive for ad hoc networks. Source authentication of a data packet is also different from the authentication of a routing packet. A secure routing protocol allows a forwarding node to buffer routing packets and then verify them later [25]. However, in order to limit the impacts of injected packets, a good forwarding node should be able to verify a received packet before forwarding it to the next hop. Hence, hop-by-hop source authentication [32, 35] has been considered as the baseline in our study to ensure that an injected false date packet can be filtered out immediately.

In hop-by-hop source authentication, the source first shares a pairwise key with each en route node according to key management protocols [4, 16, 36]. Then, the source computes an authentication token for each en route node with the key shared between them, when it needs to send a data packet. Thus, the data packet can be verified hop by hop. This approach can provide immediate source authentication and inherently supports the on-demand nature of ad hoc networks. Nevertheless, There are many practical challenges in applying this type of source authentication in an ad hoc network. The most critical issue is the unreliability of the network. For example, route change make it impossible for en route nodes to verify the source. As a result, even good packets will be discarded when these approaches are adopted for defense purposes. To address these problems, we propose a lightweight, on-demand and hop-by-hop source authentication forwarding (SAF) protocol in forwarding data packets.

Contributions The SAF protocol is specially designed to handle various problems in the forwarding procedure in an unreliable ad hoc network. In this protocol, we propose a new authentication scheme to allow en route nodes to 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. Second, 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.

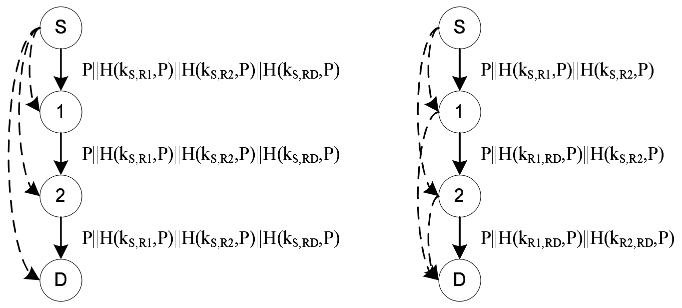
The rest of the paper is organized as follows. Section 2 presents related works on DoS research and source authentication. Section 3 presents the attack model and various problems that an authentication protocol will face. Section 4 presents the design of SAF. Its security properties are analyzed in Section 5. SAF is evaluated in Section 6. The paper is concluded in Section 7.

2 Related Works

2.1 DoS in Wireless Networks

Many approaches have been identified to launch DoS attacks in an ad hoc networks. In the physical layer, jamming [26] can disrupt and suppress normal transmission. In the MAC layer, the defects of MAC protocol messages and procedures of a MAC protocol can exploited by attackers. In the 802.11 MAC protocol, Bellardo *et al.* [2] discussed vulnerabilities on authentication and carrier sense, and showed that the attackers can provide bogus duration information or misuse the carrier sense mechanism to deceive normal nodes to avoid collision or keep silent. Gu *et al.* [8] analyzed how attackers can use certain packet generation and transmission behavior to obtain more bandwidth than other normal nodes. Wullems *et al.* [30] identified that the current implementation of the MAC protocol in the commodity wireless network cards enables an attacker to deceive other nodes to stop transmission. Researchers [1, 9, 10, 19] also found that attackers can manipulate routing procedures to break valid routes and connections. In order to prevent attackers from exploiting the security flaws in routing protocols, several secure routing protocols have been proposed to protect the routing messages, and thus prevent DoS attacks. Dahill *et al.* [28] proposed to use asymmetric cryptography for securing ad hoc routing protocols. Papadimitratos and Hass [21] proposed a routing discovery protocol that assumes a security association (SA) between a source and a destination, whereas the intermediate nodes are not authenticated. Hu, Perrig and Johnson designed SEAD [11] which uses one-way hash chains for securing DSDV, and Ariadne [9] which uses TESLA and HMAC for securing DSR. Aad *et al.* identified the JellyFish attacks that drop, reorder or delay TCP packets to disrupt TCP connections [1]. They believed that the DoS resilience relies on end-to-end detection mechanisms, because current intrusion detection approaches cannot effectively identify the attackers in ad hoc networks.

Intrusion detection is limited in ad hoc networks. Zhang *el al.* [34] proposed a general architecture to have all nodes participate in intrusion detection. Each node takes two roles. A node needs to monitor transmission in its neighborhood in order to detect misbehavior in its nearby nodes. Then, each node can cooperate with its neighboring nodes to exchange intrusion detection information in order to detect the malicious node. Marti *et al.* [17] proposed to use watchdog to detect the attacking nodes. Basically, a good node overhears its next hop to check whether its next hop forwards the packets that are received from the good node. After detecting malicious nodes, the good node uses a pathrater to exclude the malicious node from its routes. In a clustered ad hoc network, a cluster head is elected for monitoring data traffic within the transmission range [12]. All these intrusion detection approaches need nodes to monitor the transmission in their neighboring areas. However, a malicious node may use a directional antenna for transmission in order to avoid monitoring. Also, a malicious node may ask other malicious nodes to circumvent its transmission area. Hence, monitoring of nearby transmission may not be realistic in this kind of adversary environment. Furthermore, the detection relies on trusted neighboring nodes. A trusted node will honestly report misbehavior. However, a malicious node can ask another neighboring node to lie and deceive defenders.



(a) No Interleave (b) Interleave

Figure 1: Hop-by-hop authentication in packet forwarding

2.2 Source Authentication

Source authentication is mostly used to ensure that a packet comes from the claimed source. When a source sends packets to a destination, it puts authentication information into packets. A receiving node only accepts a packet if it is authenticated. In this way, only the packets from the real source can go through the route and reach the destination. Other than public key based digital signature which has unbearable computational demand on mobile nodes, several source authentication approaches exist in the literature. Multicast source authentication allows multiple receivers to verify whether the received data was originated from the claimed source and was not modified en route. Multicast source authentication amortizes the cost of a digital signature over multiple packets. Some researches proposed techniques that do tolerate packet loss in multicast source authentication by using expanded graph [29], authentication chain [6], distillation code [15] or erasure code [22]. In general, they can only tolerate the loss of a few packets. However, a node may discard all packets in its routing buffer when it is turned down in an ad hoc network. Previous approaches are unable to sustain authentication in this situation. Perrig *et al.* proposed TESLA [24] based on one-way key chain. To start the scheme, a sender uses a regular signature scheme to sign the initial key. All subsequent packets are authenticated through a one-way key chain. TESLA is efficient in computation and can tolerate the loss in the following data packets. However, this scheme requires en route nodes to buffer packets in order to verify them later. Hence, it cannot filter an injected packet instantly before it is forwarded to the next hop.

Hop-by-hop source authentication has been considered as the necessary measure to ensure that an injected data packet can be filtered out quickly. Ye *et al.* [32] proposed a statistical filtering scheme that allows en route nodes to filter out false data packets with some probability. Zhu *et al.* [35] proposed an interleaved hop-by-hop authentication scheme that guarantees that false data packets will be detected and dropped within a certain number of hops. The defense takes three steps. First, the source and the destination need to establish a route. Then, the source node sets up pairwise keys with the en route nodes. Finally, the source computes the authentication header that consists of several tokens. Each token is computed with one pairwise key so that only the node that has the pairwise key can verify the token.

When the two schemes are applied in ad hoc networks, the difference between the two schemes can be depicted in Figure 1, where a route has three hops from source S to destination D. The solid lines show the route, and the dashed lines represent how each node sets up keys. In Figure 1(a), the source sets up pairwise keys with en route nodes, but no key is set up between any two en route nodes [32]. In Figure 1(b), all nodes have an interleave association [35]. For example, when S sends a packet, it computes two message authentication codes for nodes 1 and 2. Upon receiving packets, nodes 1 and 2 recompute the message authentication codes for node D.

Due to the unreliability in ad hoc networks, a forwarding node may not be able to verify a received packet when a route is changed. Hence, we propose a scheme to improve these schemes.

3 Background

3.1 Packet Injection Attacks

A node launches packet injection attacks because it has been compromised or it intentionally does it; we do not distinguish the attack motivation here. The attacker may use its own ID, a fabricated ID, or another node ID as the source of the packets that it is injecting. We assume, however, that attackers will impersonate other non-compromised nodes to hide themselves, because it is risky for an attacker to misbehave in its own name. Figure 2 depicts a typical attack scenario. In this scenario, an attacker R_a stays in a route from R_b to R_d and exploits this route for attack. The attacker injects packets with the source address as R_b and the destination address as R_d. It can claim that all injected packets are forwarded from R_b. Without authentication, en route nodes will forward the packets. When the target traces the injected packets back along the route, the trace will go back to R_b instead of the attacker R_a in the middle.

Figure 2: Packet injection scenario

3.2 Problems in Defense

When applying source authentication in an ad hoc network, the unreliable mobile environment brings many limits on defense approaches. The corresponding problems can even be exploited by injectors to launch attacks and hide their identities.

3.2.1 Packet Loss

A packet could be lost due to communication error, hardware error, buffer overflow, etc. In a TCP session, this will trigger the source to retransmit the lost packets. However, retransmission allows attackers to legally replay packets. When attackers replay packets, these packets will be verified successfully by other en route nodes, since the replayed packets are authentic and attackers can claim that they are just retransmitting these packets. Furthermore, in some authentication approaches (for example, multicast authentication [32, 24]), authentication headers can be verified by all nodes in the network (for data integrity purposes). The attackers could thus replay these packets in other areas in the network instead of the target area or routes.

3.2.2 Route Change

In an ad hoc network, a new route may be set up for a variety of reasons. For example, the routing protocol itself enables an en route node to overhear routing messages and discover shorter routes, or the route can be broken due to link failures or the leaving of an en route node. However, if the new route diverges from the previous one, authentication in the new route will fail. Figure 3 depicts an example where the old route (solid lines) between S and D is broken at the link between nodes 2 and 3. Since node 2 knows another route (dashed lines) that can reach D, the new route diverges from the previous one at node 2. Note that nodes 3, 4 and 5 can still use the old route to forward packets, since the old route is still valid at their positions and their buffered packets have valid authentication headers.

Furthermore, there is always a latency before rediscovering a new route, because the routing protocol needs to deliver the routing error or the new route information to S. During the rediscovering period, two types of packets will be affected. One is the packets that S is sending out before S receives the routing error. The other type is the packets that nodes 1 and 2 have received from S and are trying to forward. Because S does not know the new route yet, these packets have out-of-date authentication headers. Hence, nodes 6 to 9 will not be able to verify these packets. Even after S starts a new forwarding procedure in the new route, these packets will be discarded by nodes 6 to 9, because these packets will not be delivered back to the source for re-authentication.

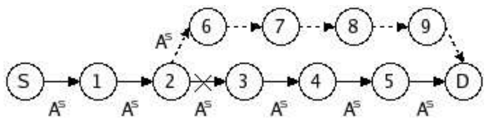


Figure 3: Change of a route

4 Design of SAF

4.1 Network and Communication Assumptions

In this study, we mainly prevent attacks in unicast communication. We assume that a failed link can trigger a node to re-discover a route. These assumptions hold in IEEE 802.11 protocol [13] and ad hoc routing protocols [14]. We consider an unreliable and mobile environment in ad hoc networks. SAF is designed to fi t in such an unpredictable and unfriendly environment. SAF is designed to work with the routing protocol DSR [14], since it needs the IDs (i.e. the node's address) of en route nodes along the forwarding path to ensure that packets will only be delivered in the claimed route. Other protocols can be extended to carry the IDs of en route nodes in order to work with our protocol. For example, in the AODV protocol [23], when a route is discovered, the destination should send a RREP back to the source along the route. En route nodes can append its ID to this RREP packet. This could work as an option in the AODV protocol when SAF is enforced. However, in this paper, we only deploy DSR for discussion.

4.2 Pairwise Keys Establishment

Hop-by-hop source authentication requires that a source node sets up a pairwise key with every en route node along the route. Because the source node can obtain IDs of en route nodes from DSR route reply packets, the source node and any one of the routing nodes can mutually figure out a pairwise key based on their IDs. Note that two en route nodes do not need to have a pairwise key.

The literature provides many key management schemes. For example, the simplest way to set up pairwise key is to pre-load pairwise keys into nodes, although it is not practical for a large and dynamic network. Novel key management schemes with better performance have also been proposed for ad hoc and sensor networks. For example, we can employ either the Blom scheme [27] or the Blundo scheme [3] directly, or use their extensions [4, 16] to tolerate a possibly larger number of node compromises. These schemes are ID-based in that two nodes only need to know each other's id to establish a pairwise key without exchanging any other information. Once a route is discovered, the source node can instantly compute pairwise keys with en route nodes based on their IDs, and start sending and forwarding packets. No addition key management packets are needed, and the storage demand and the computational overhead are shown to be affordable to current generation ad hoc nodes. A brief description of a pairwise key setting up example is provided in Appendix B.

In this study, the proposed hop-by-hop source authentication protocol is based on these existing works for key setup and management as long as they can ensure the security of the pairwise keys. This protocol focuses on solving the unreliability problems in the forwarding procedures.

4.3 Framework of SAF

Every node in an ad hoc network enforces the proposed protocol as shown in Figure 4, where the left module represents a regular or secure routing protocol, and the right module is our scheme for forwarding. The forwarding module, like the routing module, is an independent module in the network layer and decides if a data packet should be forwarded or not. Note that data packets refer to the packets in the network layer, but exclude routing packets (for routing) and keying packets (for pairwise key management). The excluded types of packets are generally secured by their own protocols [9, 11, 33, 4, 16, 36], which can prevent attackers from exploiting these packets for attack.

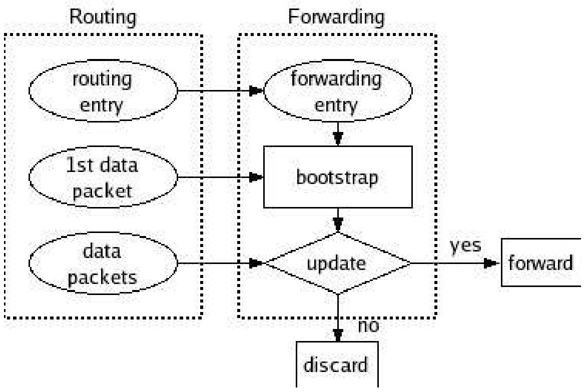


Figure 4: Framework of SAF

For discussion, we assume that a source node S sends packets 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.

4.3.1 Forwarding Entry

When S wants to send data packets to D, it uses a routing protocol to find a route. According to the routing protocol (such as DSR), each en rout node will record the source address S, the destination address D, and the routing sequence number RID in a routing entry. After the route is discovered, SAF will ask each en route node to create a forwarding entry as an extension to the routing entry. The forwarding entry will include the following information for packet verification in the forwarding procedure.

- SID: identification of source/starter,
- FID: identification of forwarding entry,
- PC_{1st}: the count of the first received packet,
- PC_{last}: the count of the last received packet.

4.3.2 Bootstrap

The bootstrap procedure is used for en route nodes to create the corresponding forwarding entry. Upon the setup of a route, the source node sends its first data packet $PKT(1)$. The source attaches an initial authentication header $A(1)$ to the packet.

$$A(1) = [SID||RID||FID||PC(1)||\delta_{R_1}(1)||...||\delta_{R_n}(1)]$$

SID is the source ID, RID is the routing sequence number, FID is the identification of the forwarding entry, and $PC(1)$ is the count of the first packet.

$\delta_{R_j}(1)$ is the authentication token for R_j. The size of an authentication token is determined by the tradeoff between security and performance. For discussion, we set a token as an 8-bit number in this study, although the hash output could be 256 bits or longer.

$$\delta_{R_j}(1) = H_{k_{SID,R_j}}(RID||FID||PC(1)||L_j)$$

k_{SID,R_j} is the pairwise key shared only between SID and R_j, and $H_k(*)$ is a keyed hash function. L_j is the sum of the data size, the number of authentication headers, and the number of remaining authentication tokens in the authentication header when the packet arrives at R_j. For example, in Figure 5, when R_2 receives a packet, the packet should include 1 authentication header, and the header has 2 tokens ($\delta_{R_2}(1)$ and $\delta_{R_3}(1)$). Hence, assume the packet has 100-byte data, then $L_2 = 100 + 1 + 2 = 103$. Similarly, when R_3 receives the packet, R_3 should have $L_3 = 100 + 1 + 1 = 102$.

Upon receiving the bootstrap packet, R_j first obtains SID, RID, FID and $PC(1)$ from the authentication header. Since R_j is in the route, R_j should be able to identify a routing entry that has S, D and RID, and thus R_j knows L_j. R_j can then verify $\delta_{R_j}(1)$. If the verification fails, the packet is discarded. If the verification is successful, R_j removes tokens (if any) for current and previous hops from $A(1)$ to save communication overhead

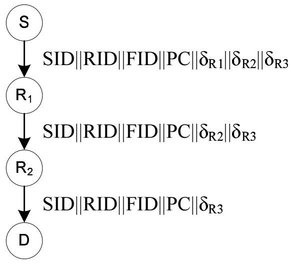


Figure 5: An example of forwarding

(because $\delta_{R_j}(1)$ and all previous tokens are no longer useful for the following en route nodes to do verification), and then forwards the bootstrap packet to the next hop R_{j+1}.

Every en route node R_j will create a new forwarding entry to record SID and FID and keep two copies of $PC(1)$ in the new forwarding entry. One copy of $PC(1)$ indicates the packet count of the first received packet, still denoted as PC_{1st}. The other copy indicates the packet count of the last received packet, denoted as PC_{last}. Note that when R_j receives $A(1)$, it will find that either no routing entry exists for S, D and RID, or no forwarding entry exists for SID and FID. Hence, the bootstrap packet is in fact the first packet that the en route node receives from the source. In this way, even the real bootstrap is lost in forwarding, each en route node can still have a valid forwarding entry bootstrapped by the first packet that the node receives later. Only after SID, FID, PC_{1st} and PC_{last} are all stored in the forwarding entry, this entry is bootstrapped for later forwarding and verification.

4.3.3 Update

For each new data packet $PKT(i)$, S composes a new authentication header $A(i)$ as

$$A(i) = [SID||RID||FID||PC(i)||\delta_{R_1}(i)||...||\delta_{R_n}(i)] \tag{1}$$

$PC(i)$ is one unit increment of $PC(i-1)$, i.e. $PC(i) \leftarrow PC(i-1)+1$. $\delta_{R_j}(i)$ is computed as follows.

$$\delta_{R_j}(i) = H_{k_{SID,R_j}}(RID||FID||PC(i)||L_j) \tag{2}$$

Upon receiving $PKT(i)$, R_j first obtains S, D, SID, RID, FID and $PC(i)$ from the packet, and finds the corresponding routing and forwarding entry. R_j verifies $A(i)$ and compares $PC(i)$ with PC_{last} in the forwarding entry. If the verification is successful and $PC(i)$ is greater than the last PC_{last}, R_j updates $PC_{last} = PC(i)$ and removes $\delta_{R_j}(i)$ and all tokens (if any) for previous hops from $A(i)$. Then, R_j forwards the data packet to the next hop R_{j+1}. Otherwise, i.e. the verification fails or $PC(i) \leq PC_{last}$, R_j discards the data packet.

4.4 Forwarding in an Unreliable Ad Hoc Network

4.4.1 Solutions for Unreliability

Unreliability of an ad hoc network requires the forwarding module to handle various problems.

Packet Loss A packet could be lost due to communication error, hardware error, buffer overflow, etc. If the bootstrap packet is lost at R_j, en route nodes will treat the first received data packet as the bootstrap packet to create the corresponding forwarding entry. If a packet is lost, the forwarding module will work as follows. Assume R_j successfully receives $PKT(i)$ and updates $PC_{last} = PC(i)$, but the next several packets are lost until R_j successfully receives $PKT(i')$. R_j will check whether $PC(i') > PC_{last}$ and verify $A(i')$.

Route Change As discussed before, a route change will make en route nodes unable to verify packets and thus drop packet to cause denial of service to legitimate traffic. The idea to solve this problem is to let an en route node start another forwarding procedure in the new route. Assume the new route diverges from the old route at an en route node R_j. R_j first computes a new authentication header for each data packet as if it was the source of the new route. Then, R_j appends the new authentication header to the old header, and forwards the packet to the next node in the new route. R_j is thus called **starter**. Upon receiving a data packet, nodes in the new route verify the new header first. If the new route overlaps with the old route in some segments, nodes in the overlapping segments can also verify the old headers.

Packet Disorder In the forwarding procedure, PC is increased for every data packet, and an en route node only accepts a data packet with PC larger than the previous one. However, when a route is changed, the order of packets may be mixed or reversed. This problem happens to the nodes in the overlapping segments of the old and the new routes. It is possible that packets in the new route (having larger PC) come earlier than packets in the old route (having smaller PC). The consequence is that the data packet with smaller PC will be discarded. To solve this problem, SAF asks each forwarding node to record different PC_{1st} and PC_{last} for each route, and compare PC only with the PC_{last} corresponding to the route the packet is forwarded from.

4.4.2 Forwarding Algorithm

The forwarding algorithm has two components. The starter uses Algorithm 1 to compute authentication headers in packets, and en route nodes use Algorithm 2 to verify authentication headers in packets. An example of SAF is given in Appendix C.

Starter/Source Algorithm 1 allows a starter to add a new authentication header in a packet when the packet cannot be delivered due to route change as described in Section 4.4.1. The algorithm consists of 5 phases.

In phase 1, the starter checks whether it is the source. Differing from other starters, the source node needs to set the packet count in a packet. If the packet is not the first packet that the source sends to the destination, the source should increase the packet count by one for each new packet. All other nodes (including other starters) simply record the packet count if the packet is authenticated.

In phase 2, the starter removes the last authentication header in a packet if the header was created by the starter itself. When the starter forwards a packet to the next hop, it is possible that the link to the next hop fails. In such a situation, the packet will be returned to the starter for retransmission in a new route. The returned packet has a header created by the starter. Hence, the starter needs to replace the old header with a new one.

Phase 3 is to discover a valid route for the packet. In DSR, due to a link failure to the next hop, a route could be revoked. It is also possible that the starter receives a route error message and revokes a route. Hence, if no route is available for the packet, the starter needs to discover a new route. In DSR, the routing packets will carry S, D and RID so that all en route nodes have the corresponding information of the new route.

In phase 4, the starter creates or updates the corresponding forwarding entry. If the packet is the first packet to be delivered in the new route, the starters need to create a new forwarding entry. FID is used to uniquely identify the entry. If the forwarding entry has already been created, the starter simply records the current packet count.

Finally, in phase 5, the starter computes an authentication header and tokens as described in Eqs.(1) and (2). Then, the starter sends the packet to the next hop.

Algorithm 1 SAF in a Starter

Assume the starter SID receives a packet PKT that should be sent from S to D. Assume the packet has m authentication headers $A^1 \cdots A^m$, where A^m is the authentication header for the latest route segment the packet will go through.

set SID in PKT to be the ID of the current node;
if $SID = S$ **then** ▷ ==Phase 1==
    if this is the first packet sent from S to D **then**
        set PC in PKT to be 1;
    else
        set PC in PKT to be a one-unit increment of PC in the previous packet;
    end if
end if
set $m' = m + 1$; ▷ ==Phase 2==
if SID in A^m is the ID of the current node **then**
    remove A^m;
    set $m' = m$;
end if
if there is no valid route from S to D **then** ▷ ==Phase 3==
    find a new route to D
    create a routing entry that records the route and S, D, RID;
end if
set S, D and RID in PKT;
if there is no valid forwarding entry **then** ▷ ==Phase 4==
    create a forwarding entry F with a unique FID;
end if
set FID in PKT;
if there is no first packet count in the forwarding entry **then**
    set the first packet count in F to be PC in PKT;
end if
set PC_{last} in F to be PC in PKT; ▷ ==Phase 5==
compute and append a new authentication header $A^{m'}$ to PKT;
forward PKT to the next hop;

En Route Nodes Algorithm 2 describes how an en route node verifies a received packet. It consists of 4 phases.

In phase 1, the en route node verifies the token in the last authentication header. As discussed in Section 4.4.1, the node could receive packets from the same source via different routes. However it is obvious that the last authentication header reveals the route through which the packet goes. Hence, the node verifies whether the received packet is legitimate in the route it goes through. If it is not, the node discards it.

Then, in phase 2, the en route node checks whether the received packet is the first one sent by the starter. If the packet is the first from the starter, the en route node should have no forwarding entry for the packet yet. Thus, the node creates a forwarding entry to record corresponding information as discussed in Section 4.3.2. Otherwise, the node checks the forwarding entry to see whether or not its PC is larger than the previous one. If the PC is smaller, the packet is replayed or forged and should be discarded. After passing the first two phases, the packet should be authenticated for the current route. The en route node removes the tokens for the current and the previous hops in the last authentication header.

In phase 3, the en route node checks other authentication headers. A packet may carry multiple authentication headers, each of which represents a possible route. For each authentication header, the node checks whether or not a forwarding entry exists for verification. If an entry exists, the node verifies the token in the authentication header and makes sure PC is not in the range of the entry. If verification fails, the packet will be discarded.

Finally, in phase 4, a packet has passed all verifications, and the en route node sends the packet to the next hop.

Algorithm 2 SAF in an En Route Node

Assume an en route node receives a packet PKT that should be sent from S to D. Assume the packet has m authentication headers $A^1 \cdots A^m$, where A^m is the authentication header for the latest route segment the packet will go through.

obtain S, D, SID^m, RID^m, FID^m and PC from A^m;
find the routing entry R^n in the node according to S, D and RID^m;
verify the token for the current node in A^m; ▷ ==Phase 1==
if verification fails **then**
    discard the packet and quit;
end if
find the forwarding entry F^m in the node according to SID^m and FID^m; ▷ ==Phase 2==
if F^m does not exist **then**
    add a forwarding entry F^m;
    record SID^m and FID^m in F^m;
    record the first packet count $PC^m_{1st} = PC$ in F^m;
else if $PC \leq PC_{last}$ in F^m **then**
    discard the packet and quit;
end if
remove tokens for current and previous hops in A^m;
for i=1; $i \leq m-1$; i++ **do** ▷ ==Phase 3==
    obtain S, D, SID^i, RID^i and FID^i from A^i;
    find a forwarding entry F^i in the node according to S, D, RID^i and FID^i;
    if F^i exists **then**
        verify the token for the current node in A^i;
        if verification is not successful **then**
            discard this packet and quit;
        end if
        if $PC > PC^i_{1st} > PC^m_{1st}$ **then**
            discard this packet and quit;
        end if
        remove tokens for current and previous hops in A^i;
    end if
end for
set PC^m_{last} in F^m to be PC in PKT; ▷ ==Phase 4==
forward PKT to the next hop;

5 Security Analysis

5.1 Packet Injection

It is possible that an attacker intends to "legally" inject junk packets into the network by using its own identity. Although action can be taken to stop the injection later, we cannot prevent such a "legal" injection. The objective of this study is to force any attacker to expose its ID if it wants to inject or to quickly filter the junk packets if it impersonates other nodes. Nevertheless, according to DSR, it is quite easy to trace the attacking source when the ID is exposed. Hence, in the following security analysis, we do not consider an attacker or its coalition as a "legal" source that uses the true identity. Nevertheless, an attacker or its coalition could be a starter or an en route node.

Property 1 *If an attacker is an en route node, it is infeasible for the attacker to break tokens.*

Although a token is only a few bits of the hash output, the attacker does not know the pairwise key that is only shared between the starter and the corresponding en route node. Because the security is determined by the length of key instead of the length of token, to break the token without knowing the pairwise key is as difficult as to break the hash function. For instance, assume a key has 128 bits, and the attacker has buffered a large table of packets that exhaust all possible 8-bit tokens. As long as the keyed hash function is secure in terms of randomness and one-way, the attacker cannot derive a valid authentication token for a new packet that is not in the table. The attacker may use known plaintext cryptanalysis to crack the key. However, he has to exhaust the key space instead of the 8-bit token space, i.e. he needs to try 2^{-127} keys in average to get the right key. Hence, a large key will provide sufficient security.

Property 2 *If an attacker is an en route node, it cannot forge tokens for junk packets or replay legitimate packets.*

In order to inject a packet, the attacker needs to create a valid authentication token for either a forged packet or a replayed packet. A replayed packet can be easily detected and discarded, while a forged packet has new information. Hence, it is hard to compute a valid authentication header over the new information in the forged packet without knowing the key.

Property 3 *If an attacker is an en route node, it cannot insert junk bits into legitimate packets.*

Since the data size, the authentication header size and the number of previous authentication headers of a packet are secured in tokens, an attacker, as an en route node, cannot compute valid tokens for junk bits that are inserted into packets.

Property 4 *If an attacker is an en route node, the probability that a forged packet can survive is negligible.*

Since it is infeasible for an attacker to compute valid tokens, the attacker may try to fabricate tokens. Assume a token has l bits, the attacker has a 1 in 2^l chance to fabricate a correct token. For example, if we use 8-bit tokens, the probability that a forged packet will be accepted by the next hop is $\frac{1}{256}$. The attacker may also try a few tokens. With an increasing probability, approximately $\frac{n}{2^1}$, the attacker may succeed in sending out one packet to the next hop at the n-th attempt. However, the successful attempt does not give any clue for the attacker to forge the next injected packet, because a guessed token does not help the attacker break the authentication key. In addition, frequent authentication failure is a clear intrusion signal. Further actions can be taken to counteract upon such an event (although the paper itself does not discuss on how to take action upon detecting frequent authentication failure).

Property 5 *If an attacker claims to be a starter, it must expose its own ID to inject junk packets or insert junk bits into legitimate packets.*

Because only a starter knows the pairwise keys that are shared between itself and the corresponding en route node, it is impossible for an attacker to impersonate another node as a starter to forward packets. An attacker may claim that a route is broken and it needs a new route to forward packets. By doing so, the attacker becomes a starter to inject junk packets or insert junk bits into legitimate packets. However, the attacker needs to authenticate packets that will be forwarded in the new route. Hence, the attacker's ID will be included in the authentication tokens. Accordingly, although the attacker can "legally" inject junk packets in the new route, it cannot hide itself or impersonate another one. It is also possible that a malicious starter colludes with an en route node and let the en route node to inject. Although the en route node can hide itself, the injection will expose its partner (i.e. the colluding starter).

5.2 Misuse of SAF

An attacker may misuse SAF to cause other attacks. As an en route node, the attacker can drop, replay, disorder or modify the authentication headers in the packets that it needs to forward. Nevertheless, we find that misuse of SAF generally results in the drop of misused data packets, but does not affect other legitimate data packets. Misuse is against the objective of packet injection attacks in terms of congestion.

Property 6 *If an attacker intentionally modifies the authentication header, the result is the same as that the attacker drops the packet.*

An attacker can modify any field in the authentication headers. The modification will easily fail verification and the modified packet will be discarded. Hence, the impact of modification is the same as the drop of the modified packet. Furthermore, as discussed in Section 4.4, if the bootstrap packet or any following packet is dropped, SAF is not affected.

Property 7 *If an attacker replays a packet in other routes, the packet will be discarded.*

An attacker may replay packets in order to inject junk packets in other routes of the network. Because authentication tokens are only computed for the starter and the nodes in one route, any other node outside the route cannot verify the packet and the packet will be filtered.

Property 8 *If an attacker disorders the packets to be forwarded, the result is the same as that the attacker simply discards these disordered packets.*

Assuming that an attacker buffers a few packets, but forwards the latest packet (whose PC is the largest among all buffered packets) first and then forwards previous packets. This is how the attacker intentionally disorders the packets. A good en route node will accept the first forwarded update packet and then discard all the other buffered packets. However, sooner or later, the buffered packets will be depleted. New packets have larger PC and thus will be accepted by good en route nodes. Hence, if the attacker disorders a few packets, only these packets will be discarded.

Property 9 *If an attacker injects a high volume of forged packets into a route and its next hop has limited computation capability, the next hop may be overloaded when verifying packets.*

A forwarding node may be overloaded with a high volume of incoming packets. Due to the filtering property of SAF, the overload can only happen in one-hop. Hence, the node next to the attacking source might be overloaded. However, the attacker may exhaust the wireless bandwidth before overloading the next hop. Assume the bandwidth is B and the average size of a packet carrying an authentication header is S. If a forwarding node can verify at least $\frac{B}{S}$ packets per second, the wireless channel will be congested before the forwarding node is computationally overloaded.

5.3 Integration with Routing Protocols

SAF should be integrated with routing protocols. When the routing protocol is not secured, a route might be used for attack. In the following, we investigate three types of routings attacks that might be used for injection. In brief, SAF does not prevent, detect or mitigate any routing attack. Nevertheless, routing attacks do not help attackers inject junk packets into illegal routes, because SAF still enforces packet authentication and verification in illegal routes. Therefore, only when (a) packets are required to be delivered in a true route and (b) no junk packet is allowed to be injected in the true route, SAF should be integrated with secure routing protocols.

Route Detour Attackers can alter a route and redirect traffic in the route. For example, in a blackhole attack [9], packets can be redirected to a non-existent destination and then be dropped. Although SAF does not detect whether a route is changed, it still requires that the packets forwarded in the altered route should be authenticated. Hence, attackers cannot use altered routes for injection.

Wormhole Attack One malicious node can deliver packets via a private tunnel to another malicious node [9]. For injection, packets may be tunneled from one route to another route and then be replayed in the later route. When SAF is enforced, the replayed packets must possess valid authentication headers for the later route, so that legitimate forwarding nodes (outside the tunnel) can verify and forward the packets. Otherwise, the tunneled packets will be discarded when being forwarded in the later route. Therefore, the wormhole attack cannot be used for injection.

Sybil Attack An attacker may fabricate routing node IDs [18]. Not willing to expose its own ID, an attacker needs to use a fabricated ID as the source or starters to authenticate injected packets. However, the attacker does not hold the secrecy assigned to the fabricated ID, that is used to set up pairwise keys according to ID-based key management schemes. Hence, the fabricated IDs do not help the attacker for injection.

6 Evaluation

6.1 Simulation Settings

6.1.1 Communication Models

We implemented SAF in NS2 [20] to evaluate its performance. The simulation uses the communication model in NS2. In thephysical layer, the two-ray ground reflection model models the signal propagation. IEEE 802.11 is the MAC and PHY protocols for communication among nodes. The CSMA and DCF functions are used to avoid transmission collision among nearby nodes. Each node has a transmission range of 250 meters. The maximum bandwidth of the channel is $1Mbps$. For communications over multiple hops, DSR is used as the routing protocol. In this study, nodes are preloaded with pairwise keys, so that the evaluation of SAF will not be biased by the performance of key management. Nevertheless, we are aware that the interaction between SAF and key management schemes will affect the overall performance of secure forwarding, and the corresponding study is under going.

6.1.2 Simulation Parameters

The simulation uses the following parameters, unless otherwise mentioned. The network is in a $1500m \times 1500m$ area, and 100 nodes are randomly put in the network. Nodes move randomly at the maximum speed of $2m/s$, $5m/s$ or $10m/s$. 10 connections are set in the network. 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 $40Kbps$. The payload of a data packet is 512 bytes, and each token has 8 bits. The scenario generation tool in NS2 is used to generate various scenarios according to these parameters.

6.1.3 Performance Metrics

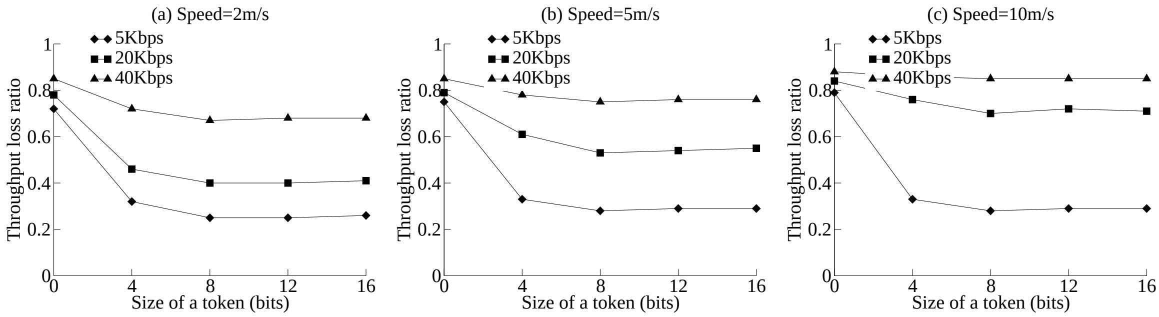
We measure five performance metrics of SAF in all scenarios. The first is *Effectiveness* (measured as the ratio of throughput loss), which shows whether SAF can filter junk packets and eliminate the attack impact. The second is *Data throughput per flow* (measured as the data rate (Kbps)), which illustrates the impact of SAF on the network. The other three metrics are used to examine how and why SAF might interfere with regular data forwarding and what cost SAF brings to the network. *Communication overhead per hop* is measured as the number of bytes that are carried to each data packet. *Authentication per starter* is measured as the number of authentication tokens that a starter computes to authenticate a data packet. *Verification per hop* is measured as the number of authentication tokens that are designated to an en route for source verification.

6.2 Evaluation Results

6.2.1 Effectiveness of SAF

SAF targets filtering injected junk packets in one hop. In this simulation, 6 attacking nodes are randomly put in the network and the attacking load of each node is $40Kbps$. They impersonate other nodes and forge tokens to inject junk packets. Because the probability that a forged token can be detected depends on the size of the token, we use Figure 6 to illustrate the effectiveness regarding various size of tokens.

First, the figure shows that the size of a token has less impact on the effectiveness of SAF when a token has more than 8 bits. When the size of a token is 8-bit, the chance to forward a forged token is only $\frac{1}{256}$. Hence, in the following simulations, we will take 8-bit as the token size. In addition, the figure shows that SAF can filter junk packet in one hope. When the network is reliable (speed is 2m/s and load is 5Kbps), SAF can reduce the throughput loss from 75% to 25%. However, the attacking nodes cannot be stopped from injecting. The attack impact still exists nearby the attacking nodes. In order to thoroughly eliminate the attack, the attacking nodes have to be physically removed from the network. We also notice that when the network is unreliable (speed is 10m/s and load is 40Kbps), SAF has limited effect on the traffic, because normal traffic suffers a high throughput loss from itself as shown in Figure 7.



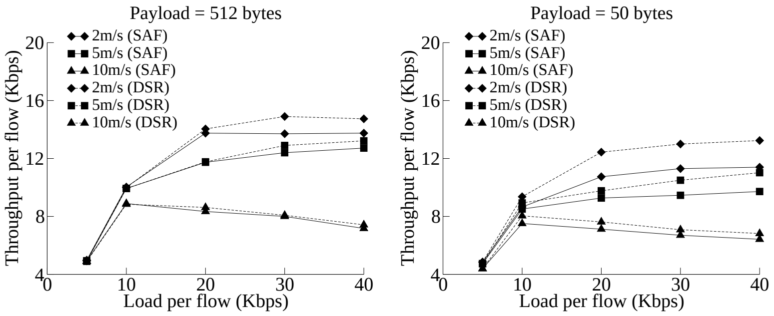
Each sub figure shows the results under different node speeds. Each curve shows the throughput loss under attack regarding different normal traffic loads and different token sizes. When the token size is 0, SAF is not enforced to protect the network.

Figure 6: Effectiveness of SAF

6.2.2 Throughput of SAF

Figure 7 is to address the major concern on whether or not SAF will affect the throughput. We conduct comparison between DSR and SAF given two types of payload. One type of payload is 50 bytes per packet, and the other is 512 bytes per packet.

As illustrated, SAF does not interfere with DSR when the packet size is large (512 bytes). Compared with the sizes of payload, IP header and MAC header, the overhead of SAF is lightweight, around 10 to 24 bytes in our simulation (as depicted in Figure 8). Only when the network is unreliable (the load is more than $30Kbps$ and the speed is $10m/s$), the throughput of SAF deviates from DSR. When the packet is small (50 bytes), the difference of throughput between SAF and DSR becomes significant. Especially, when the network is unreliable, SAF may append authentication headers that are larger than the data packet, and thus reduce the throughput. Figure 7 also shows that SAF is practical in an unreliable ad hoc network. The solid lines demonstrate that SAF can work even when around 70% of packets are dropped. Note that the network may be disrupted by the legitimate traffic, since SAF does not set any rate limit on legitimate traffic.



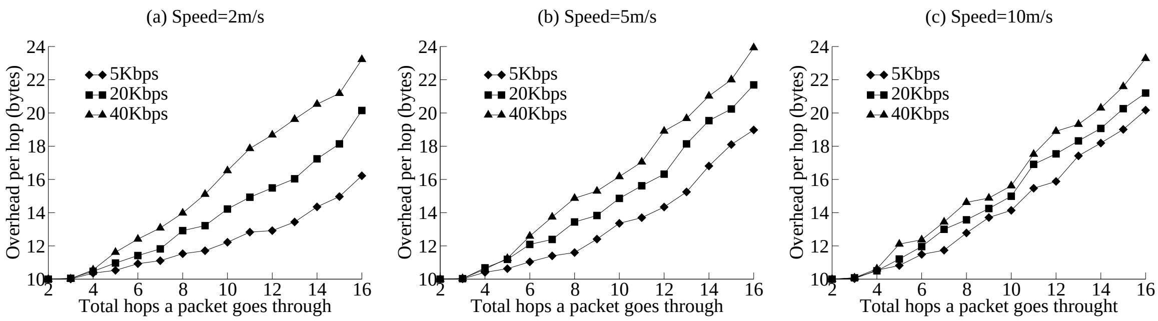
Throughput of SAF is represented by solid lines, and DSR by dashed lines. Each sub fi gure shows the throughput with different payloads. Each curve shows the throughput regarding different node speeds and different different normal traffi c loads.

Figure 7: Throughput comparison

6.2.3 Overhead of SAF

The size of authentication headers change along the route, as an en route node removes its corresponding authentication tokens from a packet when it forwards the packet, or a starter adds new authentication tokens to a data packet for the new segment in the path. Figure 8 shows the average overhead vs. the total hops of a path.

As illustrated, the authentication header is larger when the path is longer. When the destination is far away from the source or the network is unreliable, a packet has to go through several new segments in the path. The overhead has a constant part about 10 bytes, and increases linearly to the total hops with a slope that is influenced by the load and the speed. Our simulation shows that a path with one more hop adds 0.5 bytes to the average overhead when the load is light (5Kbps) and the speed is low (2m/s). On the other hand, when the load or the speed is high, the network becomes unreliable, and the overhead increases more quickly. In the unreliable environment (40Kbps and 10m/s), a path with one more hop could increase the overhead by more than 1 byte on average. Furthermore, when the speed is low, there is an obvious difference of slopes under various loads. While the speed is high, this difference is diminished.

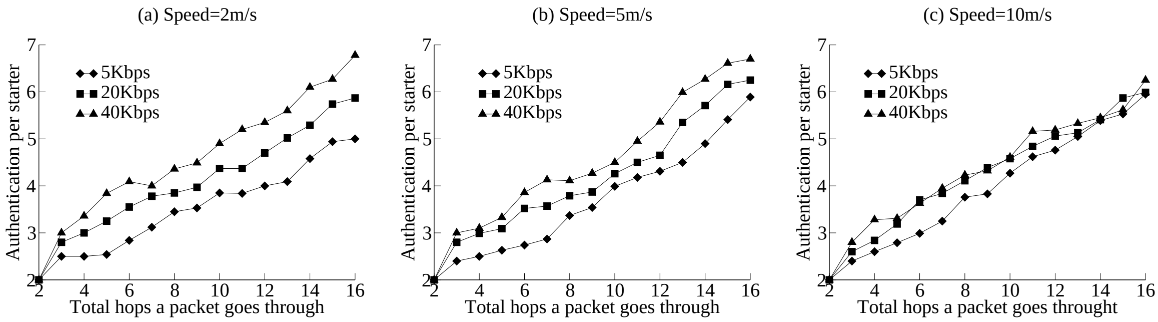


Each sub fi gure shows the overhead per hop at different node speeds, and each curve represents the overhead at different normal traffi c loads and different path lengths.

Figure 8: Communication overhead per hop

6.2.4 Computation of SAF

The starter needs to compute authentication headers for data packets and each en route node needs to verify packet sources. The computational demand for starters, measured as the number of authentication tokens that a starter needs to compute, is depicted in Figure 9. Differing from the overhead, the computation for authentication does not increase as much as overhead when the path is longer. As we trace each data packet, we find that many data packets go through a path with several new segments before reaching the destination and each new segment needs a starter to compute a new authentication header. Hence, even when the whole path is longer, each starter in the path only computes for its own segment. However, the accumulative computation of all starters along the path might increase more as the path gets longer, which can be inferred from the average overhead of the path. Similar to overhead, network unreliability (higher load and speed) increases the computation for starters (although slightly). In the worst case (40Kbps and 10m/s), a starter needs to compute around 0.3 authentication tokens on average for each hop in the path.



Each sub figure shows the computation of a starter at different node speeds, and each curve represents the computation at different normal traffic loads and different path lengths.

Figure 9: Number of authentication tokens a starter needs to compute

The computation cost for each en route node, which is measured as the number of authentication tokens the node needs to verify, is depicted in Figure 10. In fact, the per hop computation is less related to the total hops. Hence, the figure directly shows the influences of load and speed on verification. Load is 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 many new segments in a path do not overlap with the old segments. Hence, even if a data packet carries a large authentication header with many authentication tokens, each en route node may only find one or two tokens that are designated to it. In another words, many tokens for broken routes in an unreliable environment cannot be verified in the new segments, which is the reason that source authentication approaches in the literature are not suitable in ad hoc networks.

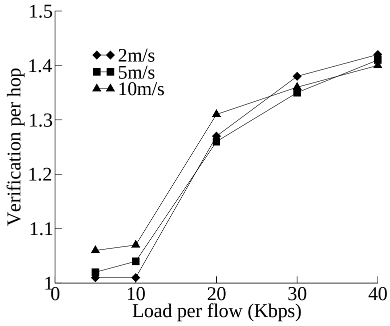


Figure 10: Number of authentication tokens a hop needs to verify

6.3 Comparison

In Table 1, we compare the major feature of SAF and two existing hop-by-hop source authentication protocols [32, 35] in terms of whether they can handle unreliability, whether they ensure security in forwarding, whether they need a key management scheme, and the size of authentication headers.

The comparison shows that the two protocols are not suitable in unreliable ad hoc networks because they cannot handle both unreliability and security at the same time. However, such a difference is due to their target applications, in which neither the packet injection attack nor the unreliability is a major concern. At the same time, the overhead of SAF is similar to the two protocols (proportional to the length of a path). But, the overhead of SAF is influenced by the unreliability of the network, which is depicted in Figure 8.

Scheme	Unreliability	Security	Key Scheme	Header size
SAF	Y	Y	Y	$O(kn)^{1,2}$
[32]	Y	N	Y	$O(n)^1$
[35]	N	Y	Y	$O(m)^3$

[1] n is the length of a path.
[2] k is a coeffi cient that varies according to the unreliability.
[3] m is the maximum number of colluding nodes in a path, and $m < n$.

Table 1: Feature comparison

7 Conclusion and Future Works

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. This protocol is designed to fi t in the unreliable environment of ad hoc networks. The protocol can either immediately fi lter out injected junk data packets with very high probability or expose the true identity of the injector. 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, interfering negligibly with regular packet forwarding.

One of the major future work is to integerate non-ID-based key management schemes with SAF, because SAF relies on a key management scheme to establish pairwise keys. A non-ID-based key management scheme may affect the overall performance of packet forwarding in several aspects. First, a non-ID-based key management scheme has its own overhead, which can interfere normal traffic in a network. Second, a non-ID-based key management scheme is affected by the characteristics of a network as normal traffic. When a network is less unreliable or nodes join and leave more frequently, pairwise keys are harder to set and maintain, and correspondingly SAF may not be able to forward packets. Besides examining SAF with various key management schemes, we can also integrate the procedure of pairwise key establishment into secure routing protocols and the bootstrap procedure in SAF. More work is needed on these issues.

References

[1] I. Aad, J.P. Hubaux, and E. Knightly. Denial of service resilience in ad hoc networks. In *ACM MobiCom*, 2004.

[2] John Bellardo and Stefan Savage. 802.11 denial-of-service attacks: real vulnerabilities and practical solutions. In *USENIX Security Symposium*, pages 15–28, Washington D.C., 2003.

[3] C. Blundo, A. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung. Perfectly-secure key distribution for dynamic conferences. In *Advances in Cryptology, CRYPTO 92, LNCS*, volume 740, pages 471–486, 1993.

[4] Wenliang Du, Jing Deng, Yunghsiang S. Han, and Pramod K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *ACM CCS*, pages 42–51, 2003.

[5] Zhenghua Fu, Petros Zerfos, Kaixin Xu, Haiyun Luo, Songwu Lu, Lixia Zhang, and Mario Gerla. The impact of multihop wireless channel on tcp throughput and loss. In *IEEE Infocom*, volume 3, pages 1744–1753, San Francisco, 2003.

[6] P. Golle and N. Modadugu. Authenticating streamed data in the presence of random packet loss. In *NDSS*, pages 13–22, 2001.

[7] Qijun Gu. *A study of selected security issues in wireless networks*. Ph.d. thesis, Pennsylvania State University, 2005.

[8] Qijun Gu, Peng Liu, and Chao-Hsien Chu. Tactical bandwidth exhaustion in ad hoc networks. In *the 5th Annual IEEE Information Assurance Workshop*, pages 257–264, West Point, NY, 2004.

[9] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: a secure on-demand routing protocol for ad hoc networks. In *ACM MobiCom*, pages 12–23, 2002.

[10] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Rushing attacks and defense in wireless ad hoc network routing protocols. In *ACM workshop on Wireless security*, pages 30–40, 2003.

[11] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Sead: secure efficient distance vector routing for mobile wireless ad hoc networks. *Ad Hoc Networks*, 1(1):175–192, 2003.

[12] Yi-An Huang and Wenke Lee. A cooperative intrusion detection system for ad hoc networks. In *the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 135–147, 2003.

[13] IEEE. Wireless lan medium access control (mac) and physical (phy) layer specification, June 1999.

[14] 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.

[15] Chris Karlof, Naveen Sastry, Yaping Li, Adrian Perrig, and Doug Tygar. Distillation codes and applications to dos resistant multicast authentication. In *NDSS*, 2004.

[16] Donggang Liu and Peng Ning. Establishing pairwise keys in distributed sensor networks. In *ACM CCS*, pages 52–61, 2003.

[17] Sergio Marti, T. J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *ACM MobiCom*, pages 255–265, Boston, Massachusetts, United States, 2000. ACM Press New York, NY, USA.

[18] J. Newsome, E. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: analysis and defenses. In *the 3rd International Symposium on Information Processing in Sensor Networks*, pages 259–268, 2004.

[19] Peng Ning and Kun Sun. How to misuse aodv: a case study of insider attacks against mobile ad-hoc routing protocols. In *the 4th Annual IEEE Information Assurance Workshop*, pages 60–67, West Point, 2003.

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

[21] P. Papadimitratos and Z.J. Haas. Secure routing for mobile ad hoc networks. In *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference*, San Antonio, TX, 2002.

[22] Jung Min Park, Edwin K. P. Chong, and Howard Jay Siegel. Efficient multicast stream authentication using erasure codes. *ACM Transactions on Information and System Security*, 6(2):258–285, 2003.

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

[24] A. Perrig, R. 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, Berkeley, CA, 2000.

[25] Adrian Perrig, Ran Canetti, Dawn Song, and Doug Tygar. Efficient and secure source authentication for multicast. In *NDSS*, 2001.

[26] Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Communications of the ACM*, 47(6):53–57, June 2004.

[27] R. Blom. An Optimal Class of Symmetric Key Generation Systems. In *Advances in Cryptology, EUROCRYPT'84, LNCS*, volume 209, pages 335–338, 1984.

[28] Kimaya Sanzgiri, Bridget Dahill, Brian Neil Levine, Clay Shields, and Elizabeth M. Belding-Royer. A secure routing protocol for ad hoc networks. In *IEEE ICNP*, pages 78–89, 2002.

[29] D. Song, D. Zuckerman, and J.D. Tygar. Expander graphs for digital stream authentication and robust overlay networks. In *IEEE Symposium on Security and Privacy*, pages 241–253, 2002.

[30] Chris Wullems, Kevin Tham, Jason Smith, and Mark Looi. Technical summary of denial of service attack against ieee 802.11 dsss based wireless lans. Technical report, Information Security Research Centre, Queensland University of Technology, Brisbane, Australia, 2004.

[31] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *ACM Mobihoc*, pages 46–57, 2005.

[32] Fan Ye, Haiyun Luo, Songwu Lu, and Lixia Zhang. Statistical en-route detection and filtering of injected false data in sensor networks. In *IEEE Infocom*, 2004.

[33] Manel Guerrero Zapata and N. Asokan. Securing ad hoc routing protocols. In *ACM workshop on Wireless Security*, pages 1–10, Atlanta, GA, USA, 2002. ACM Press New York, NY, USA.

[34] Yongguang Zhang and Wenke Lee. Intrusion detection in wireless ad-hoc networks. In *ACM MobiCom*, pages 275–283, 2000.

[35] Sencun Zhu, Sanjeev Setia, Sushil Jajodia, and Peng 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.

[36] Sencun Zhu, Shouhuai Xu, S. Setia, and S. Jajodia. Establishing pairwise keys for secure communication in ad hoc networks: a probabilistic approach. In *IEEE ICNP*, pages 326–335, 2003.

Biography

Qijun Gu is an assistant professor in Department of Computer Science, Texas State University San Marcos. He received the Ph.D. degree in Information Sciences and Technology from Pennsylvania State University in 2005, the Master degree and the Bachelor degree from Peking University, China, in 2001 and 1998. His research interests include wireless/mobile computing, denial of service, key management, ad hoc network, networking optimization, P2P sharing system.

Peng Liu is an assistant professor in School of Information Sciences and Technology, Pennsylvania State University. He received the Ph.D. degree in Information Technology from George Mason University in 1999, the Master degree and the Bachelor degree from University of Science and Technology of China in 1996 and 1993. His research interests include survivable systems, network security, database security, privacy, distributed systems security, wireless security, e-commerce, digital health care, e-government, cyber infrastructure.

Chao-Hsien Chu is an associate professor in School of Information Sciences and Technology, Pennsylvania State University. He received the Ph.D. degree in Business Administration from the Pennsylvania State University in 1984, an MBA from Tatung Institute of Technology (Taiwan), and a B.E. in Industrial Engineering (I.E.) from Chung Yuan University (Taiwan). His research interests include intelligent technologies and their applications to data mining, manufacturing systems design, information and cyber security, wireless/mobile computing, supply chain integration and management.

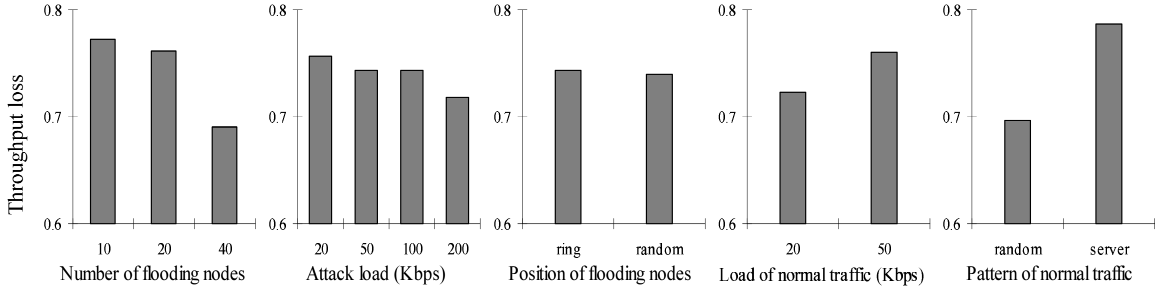


Figure 11: Impacts of injected packets in multi-hop paths

Sencun Zhu is an assistant professor in Department of Computer Science and Engineering, Pennsylvania State University. He received the PhD degree in Information Technology from George Mason University in 2004, the M.S. degree from University of Science and Technology of China in 1999, and the B.S. degree from Tsinghua University, China, in 1996. His research interests include network and systems security, ad-hoc and sensor networks, performance evaluation, peer-to-peer computing.

A Impact of Injection Attacks

In our previous study [7], various injection attack strategies and consequences were evaluated in NS2 [20]. We used CBR agents to generate both legitimate and injected traffic. All injected traffic was UDP traffic, while 80% of the legitimate traffic was TCP traffic and the remaining 20% was UDP traffic. Six factors that may affect the attack consequence were considered in this study: (1) two different types of injection paths (multi-hop or one-hop); (2) four different attack loads per injecting node (20, 50, 100, or 200 Kbps); (3) three different numbers of injecting nodes (10, 20, or 40); (4) two different types of injecting nodedeployment (random deployment or ring deployment circumventing a target); (5) two different traffic loads per legitimate node (20 and 50 Kbps); and (6) two different patterns of legitimate traffic (random, or concentrated at a service point). We also studied interactions among the last five factors. Hence, an experimental design with 192 cells was used to represent the combinations of all the factors.

The throughput loss of the legitimate traffic was measured to evaluate the attack impacts. The throughput loss is defined as the percentage of the bits in all dropped legitimate packets over the total bits in all legitimate packets during the attacks. The higher the throughput loss, the less the normal traffic can reach its destination and thus the more damage the attacks cause. Each point of the throughput loss in the comparison figures is the average of the four independent simulations. Note that the throughput loss is related to many factors in the application layer, such as extra delay of legitimate traffic due to retransmission of the lost packets or disconnection of legitimate connections due to the loss of service request packets.

Figures 11 and 12 summarize an overall evaluation of the attack impacts under various factors. On average, the throughput loss (0.74 ± 0.15) caused by injection attacks in multi-hop paths is significantly higher than injection in one-hop paths (0.55 ± 0.23). Hence, we list our major findings below for injection attacks in multi-hop paths, that showed statistical significance according to ANOVA analysis. More analysis on other factors and their interaction can be found in [7].

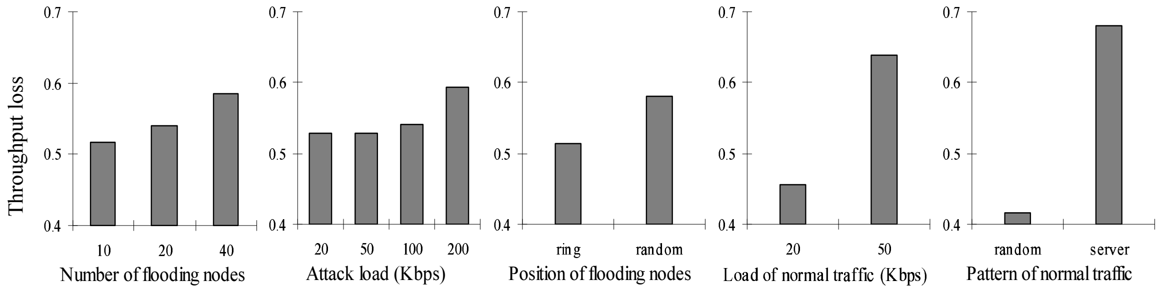


Figure 12: Impacts of injected packets in one-hop paths

- A higher load of injection traffic reduces the throughput loss, because the injected traffic cannot go through multi-hops due to problems in multi-hop transmission, such as exposed nodes and link failure [5]. Consequently, a lower attack load allows more injected packets to interfere legitimate traffic.
- More injecting nodes leads to less throughput loss. For instance, the throughput loss drops from 77% for 10 flooding nodes to 69% for 40 flooding nodes. It is due to the same reason that more injection flows interfere with each other and reduce their impacts on legitimate traffic. In our study, 10 injecting nodes, each of which injects at $20kbps$ in a multi-hop path, can achieve the largest attack impact.
- A higher load of legitimate traffic causes higher throughput loss due to self congestion. If an ad hoc network is full of legitimate traffic, the result will be similar to an injection DoS attack. From the attack viewpoint, attackers only need to deploy the flooding nodes in an area where normal traffic is not intense.

Our study indicates that the injection attack is a serious threat to ad hoc networks. The attack can significantly reduce the throughput of legitimate traffic. In particular, injection in multi-hop paths is a more effective and efficient method for DoS attackers to damage the network. The attack does not demand more attacking resource either. More injecting nodes and higher attack load cannot increase, but even reduce the attack impacts. Therefore, more research is needed to defend against injection attacks.

B Blundo Scheme

The Blundo scheme [3] was designed to allow any group of t users to compute a common key while being secure against coalitions of up to k users when the group of users hold IDs of the other group members. In this study, we reduce the scheme to allow any two nodes to set up a pairwise key based on their IDs. With this capability, the source node can set up a pairwise key with each en route node when a route is discovered.

The scheme works in the following steps. First, the key server randomly generates a symmetric bivariate k-degree polynomial function over a finite field GF_q, where q is a prime number that is large enough to accommodate a cryptographic key. $f(x;y)$ is said to be symmetric if $f(x;y) = f(y;x)$.

$$f(x;y) = \sum_{m,n=0}^{k} a_{mn} x^m y^n$$

Then, the key server computes $f(i;y)$ for node i, and loads all $c_{i,n}$ into node i before deploying it into the network.

$$f(i;y) = \sum_{m,n=0}^{k} a_{mn} i^m y^n = \sum_{n=0}^{k} c_{i,n} y^n$$

When two nodes i and j want to establish a pairwise key, they compute $f(i;j)$ and $f(j;i)$ by evaluating $f(i;y)$ with node ID j and $f(j;y)$ with node ID i, respectively. Because $f(i;j)$ equals to $f(j;i)$ due to the symmetric, $f(i;j)$ serves as their pairwise key.

The above scheme has been proved to be unconditionally secure and k-collusion resistant [3]; that is, an adversary knows nothing about the pairwise key between any two non-compromised nodes if the number of sensor nodes it has compromised is no more than k. However, if the adversary compromises more than k nodes, it will know all the pairwise keys in the network. Therefore, it is important to choose a large enough degree k for the polynomial for the application under consideration. If 256-bit pairwise keys are used and the network is expected to be resistant against up to 400 compromised nodes, about $13Kbyte$ memory is needed in each ad hoc node i to store its coeffi cients $c_{i,n}$, for $0 \leq n \leq k$. Futhermore, the computation (mainly including multiplication and modular) of the scheme is $O(k)$. Hence, the scheme can be applied in current ad hoc networks.

C Example of Handling Unreliability by SAF

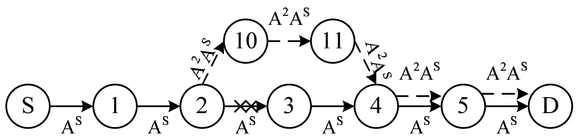


Figure 13: Forwarding in a new route, which overlaps with the old one in some segment.

In Figure 13, we assume a new route diverges from a broken route at node 2 and then overlaps with the old route at nodes 4 and 5. When node 2 receives $PKT(\alpha)$, it should see an authentication header A^S as follows

$$A^S(\alpha) = [SID^S||RID^S||FID^S||PC(\alpha)||\delta^S_{R_2}(\alpha)||...||\delta^S_{R_5}(\alpha)||\delta^S_{R_D}(\alpha)]$$

Where $*^S$ means the information from the the source S, and

$$\delta^S_{R_j}(\alpha) = H_{k_{SID^S,R_j}}(RID^S||FID^S||PC(\alpha)||L^S_j)$$

Assume the packet has 100-byte data. Because the packet has only one authentication header and there should be 5 tokens when node 2 receives it, $L^S_2 = 100+1+5 = 106$. If node 3 can receive the packet, $L^S_3 = 100+1+4 = 105$.

Assume the old route is broken when node 2 tries to forward $PKT(\alpha)$ to node 3. Now, node 2 appends a new authentication header A^2 to the authentication header A^S in each data packet. Node 2 computes A^2 as if node 2 was the source of the new route, and thus node 2 is the starter of the new route.

$$A^2(\alpha) = [SID^2||FID^2||PC(\alpha)||\delta^2_{R_6}(\alpha)||...||\delta^2_{R_9}(\alpha)||\delta^2_{R_D}(\alpha)]$$

Where $*^2$ means the information from node 2, and

$$\delta^2_{R_j}(\alpha) = H_{k_{SID^2,R_j}}(RID^2||FID^2||PC(\alpha)||L^2_j)$$

Still assume the packet has 100-byte data. Because the packet has two authentication headers and there should be 5 tokens in A^2 when node 10 receives it, $L^2_{10} = 100 + 2 + 5 = 107$. When node 11 receives the packet, $L^2_{11} = 100 + 2 + 4 = 106$.

Node 2 appends A^2 to A^S. Hence, node 10 and all following nodes in the new route will see two authentication headers in packets. Because they can verify A^2, they will not discard packets in the new route. Note that node 1 may not have any information about the new route and do not have any information of the new forwarding procedure in the new route. Node 1 may work as if nothing happens in the route. This new forwarding procedure works until S knows the new route and resets forwarding.

Assume that node 3 is congested for a long time after the new route is discovered. Hence, the packets going through node 11 will reach node 4 before the old packets buffered in node 3. Because the packets buffered in node 3 have smaller PC, they will be discarded by node 4 if node 4 only records the latest PC in the packets from node 11. According to SAF, node 4 actually has created two forwarding entries for A^2 and A^S. In this two entries, node 4 records two packet counts: $PC^S_{1st} = PC(1)$ and $PC^2_{1st} = PC(\alpha)$ (assume the new route is set up by node 2 when forwarding $PKT(\alpha)$). Obviously, $PC(i')$ in any packet buffered in node 3 satisfies $PC^S_{1st} < PC(i') < PC^2_{1st}$; while $PC(i)$ in any packet going through node 11 satisfies $PC^2_{1st} < PC(i)$. Node 4 also records two PC_{last} for the two entries respectively, denoted as PC^S_{last} for the entry that has $SID = S$ and PC^2_{last} for the entry that has $SID = 2$. When node 4 receives an packet $PKT(i)$, it compares $PC(i)$ with PC^S_{last} if the packet comes from node 3. Otherwise, it compares $PC(i)$ with PC^2_{last}.