Defending against Packet Injection Attacks in Unreliable Ad Hoc Networks

Qijun Gu, Peng Liu, Sencun Zhu, and Chao-Hsien Chu

Pennsylvania State University, University Park, PA 16802

qgu@ist.psu.edu, pliu@ist.psu.edu, szhu@cse.psu.edu, chu@ist.psu.edu

Abstract. Ad hoc networks are usually unreliable and have very limited network resources. In such a network, packet injection attacks can cause serious denial-of-service via wireless channel contention and network congestion, and are hard to defend. Although ad hoc network security has been extensively studied, most of the previous work focuses on secure routing. They cannot prevent an attacker from injecting a large number of junk packets into a route that has been established. To defend against this type of injection attack, we propose an on-demand hop-by-hop source authentication protocol in forwarding data packet. This protocol is designed to fit in the unreliable environment of ad hoc networks. The protocol can either immediately filter out injected junk data packets with very high probability or expose the true identity of the injector. It is also resistant to impersonation attacks launched by colluding inside attackers. The simulation shows that its overhead in communication and computation is lightweight.

Keywords − Packet injection, source authentication, ad hoc network, denial of service

1 Introduction

Denial-of-Service (DoS) attacks are a serious threat to ad hoc networks where the network resources are typically very limited. Since wireless channels are a shared media, injecting a large number of junk packets into a set of legitimate routes toward the target may not only cause serious network congestion at the target side, but also lead to severe wireless channel contention along each of the legitimate routes. Hence, in this paper, we are mainly concerned with this type of extra packet injection attack. Compared with other types of DoS attacks in ad hoc networks, extra packet injection attacks in general are easier to enforce, but more difficult to defend, since attackers can claim to be forwarding packets instead of changing or misusing the underlying routing and MAC protocols.

Ad hoc network security has been extensively studied. However, most of the previous work [1–3] focuses on secure routing. After a route is established, there is no authentication in forwarding data packets. As a result, an attacker, no matter he is an outsider or insider, can inject a large number of junk data packets into the route. And the resources of the intermediate (routing) nodes are greatly consumed. To mitigate this attack, an en route node needs to filter out the injected junk data packets as early as possible, not leaving it for the destination to detect. The longer time a junk data packet stays in the network, the more resources it will consume. One defense approach is to enforce source authentication in forwarding data packets. When the source sends a data packet after discovering a route, it puts authentication information into the data packet. An en route node only forwards the data packet if the packet is authentic. In this way, only the data packets from the real source can go through the route and reach the destination. Furthermore, hop-by-hop source authentication is necessary to ensure that an injected date packet can be filtered out immediately.

If some nodes do not verify authentication, an attacker may inject packets into the network through these nodes.

Among all source authentication schemes, symmetric key based approaches are a good solution [4, 5] because of their security advantages and lightweight overheads. The source 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 with the pairwise key, and thus the data packet can be verified hop by hop. This approach can provide immediate source authentication and thus inherently supports the on-demand nature of ad hoc networks. However, this approach faces many practical challenges in ad hoc networks, among which the most critical one is due to the unreliability of ad hoc networks (for example, packet loss and route change), which we will discuss in detail in this paper. To address these problems, we propose an on-demand and hop-by-hop source authentication protocol in forwarding data packets (so called **SAF**).

Our contribution in this paper is three fold. First, the proposed protocol is the first one that is specially designed to handle various problems in the forwarding procedure in an unreliable ad hoc network. It not only provides the security against packet injection attacks, but also ensures the smooth delivery of legitimate data packets. Second, the attack studied in this paper is a new type of packet injection attack, which is different from false data packet injection studied in the literature. In this attack, attackers wants to inject extra junk packets into the network instead of providing false information in the injected packets. Third, in this paper, we systematically analyzed and summarized various problems when applying source authentication in forwarding data packets in ad hoc networks. We also studied how attackers can misuse source authentication to cause other security issues.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 introduces assumptions and attack models, and then gives the basic idea of SAF. In Section 4, we analyze various situations in an unreliable ad hoc network, and discuss SAF in detail. In Section 5, we analyze its security against injection attacks and whether misuse of this protocol will cause other problems. We evaluate SAF in Section 6 and conclude in Section 7.

2 Related Works

Many previous work on ad hoc network security focused on secure routing. Dahill *et al* [6] identified several security vulnerabilities in AODV and DSR, and proposed to use asymmetric cryptography for securing ad hoc routing protocols. Yi *et al* [7] presented a security-aware routing protocol which uses security (e.g., trust level in a trust hierarchy) as the metric for route discovery between pairs. Papadimitratos *et al* [8] 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 *et al* designed SEAD [2] which uses one-way hash chains for securing DSDV, and Ariadne *et al* [1] which uses TESLA and HMAC for securing DSR.

The main difference between SAF and the above protocols is in the design goals. The above protocols are designed to secure the route discovery, whereas our scheme focuses on filtering and dropping the extra data packets injected into the routes established by those routing protocols. Therefore, our scheme and the secure routing protocols are complementary to each other.

Recently two techniques [4, 5] have been proposed for filtering injected false data packets in sensor networks. In [4], Ye *et al* propose 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 an extra packet can go through the network (although it will be discarded at the

destination) if it does not carry keys that the en route nodes have. In [5], Zhu *et al* propose an interleaved hop-by-hop authentication scheme that guarantees that false data packets will be detected and dropped within a certain number of hops. This scheme cannot be applied in our study either, because the interleave relationship cannot sustain when the route changes.

Source authentication approaches in multicast are also candidate solutions, since they allow multiple receivers to verify the received packets. However, some approaches [9, 10] do not tolerate any packet losses, and some others [11–13] tolerate limited packet loss where ad hoc networks could be much worse. Furthermore, most of these approaches do not provide immediate source authentication. When they are employed in ad hoc networks, either a sender has to know in advance one or several packets that will be sent, or the en route nodes and the destination node have to buffer the received data packets until they are verifiable. In addition, these schemes mainly target at integrity. However, an injector may be able to replicate these signed packets and replay them in other area of an ad hoc network for injection purpose. As a tradeoff between security and performance, our scheme allows an en route node to filter out injected extra data packets immediately while leaving the data integrity check to the destination node.

3 Design Overview

3.1 Assumptions

Network Assumptions This paper mainly studies unicast communication. We assume that in the wireless communication, a failure link can trigger a node to re-discover a route. These assumptions hold in IEEE 802.11 protocol [14] and ad hoc routing protocols [15]. SAF is designed to work with the routing protocol DSR [15], since it needs IDs (i.e. the node's address) of en route nodes along the forwarding path. Other protocols, such as AODV [16], cannot be applied with our protocol directly, unless these protocols are extended to carry IDs of en route nodes in the routing protocol 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, and a route could be broken and changed due to power down of a routing node. SAF is designed to fit in such an unpredictable and unfriendly environment.

Security Assumptions We notice that an attacker can launch DoS attacks at the physical layer or the data link layer. For example, it can jam the radio channel or interrupt the medium access control protocol. However, this paper does not address these attacks. This paper neither addresses security attacks against routing control packets, which have been addressed by secure routing protocols [1–3].

We assume that the source node can establish a pairwise key with every routing node along the path. Because the source node can obtain IDs of routing nodes from DSR route reply packets, the source node and any one of the routing nodes can mutually figure out a pairwise secret based on their IDs. For example, they can be simply preloaded with keys paired with IDs before they enter the network. After they know the IDs, they just pick the key corresponding to the IDs. Several probabilistic key pre-deployment schemes [17–19] can also be applied to derive such a pairwise secret if memory space of a node is a concern.

Note that in SAF, an en route node does not verify whether the content of a data packet is maliciously modified, but only make sure that the data packet is originated from the claimed source by verifying the attached segment. The authentication of the packet content can be accomplished with any end-to-end authentication protocol.

3.2 Attack Models

We mainly consider the extra packet injection attacks in which an attacker injects junk packets into an ad hoc network with the goal of depleting the resources of the nodes that relay or receive the packets. The attacker could be an *outsider* (unauthorized) node that does not possess a valid credential, or an *insider* (authorized) node that possesses a valid credential. In particular, regarding a specific route, its insiders are the en route nodes which are supposed to have the credential to verify and forward data packets, and its outsiders are the nodes which are not.

A node launches resource consumption 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, fabricated IDs, or other nodes' IDs as the sources of the packets it is injecting. We assume, however, that attackers will impersonate other non-compromised nodes to hide themselves, because an attacker takes a great risk when misbehaving in its own name.

To achieve the attack goal, an attacker may eavesdrop on all traffic, replay older packets, or modify overheard packets and re-inject them into network. We further assume that multiple attackers may collude to launch attacks. For example, multiple compromised nodes in a route may attempt to impersonate the data source by sharing the confidential information the source has offered to them.

3.3 Forwarding Module

We first give the simple version of SAF in a reliable ad hoc network, where the route from a source to a destination will not break and no packet will be lost during the transmission in the network. Then, in Section 4, we present the complete version of the protocol to handle problems in a real unreliable ad hoc network.

SAF is designed for data forwarding as shown in Figure 1(a), where the left module represents a regular or secure routing protocol, and the right module is our scheme for forwarding. The forwarding module is an independent module in the network layer as a routing protocol does, and decides whether a data packet[1] should be forwarded or not. Without SAF, an en route node simply forwards packets to the next hop that it can find from its routing table according to the destination address in the packet.

The forwarding module consists of three subcomponents: *forwarding entry creation*, *forwarding bootstrap*, and *forwarding verification and update*. 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.

Entry Creation. When S wants to send data packets to D, it uses the routing protocol to find a route. The routing packets trigger the forwarding entry creation component to create a forwarding entry for the route. SAF requires that the source node knows the IDs of the en route nodes. This information is readily available in the ROUTE REPLY packet of DSR [15]. Based on IDs, S and each en route node R_j can set up a pairwise key k_{S,R_j} according to [17–19].

Bootstrap. Upon the setup of a route, the source node sends its first data packet, its forwarding bootstrap component attaches initial authentication header $A(1)$ to the packet. An en route node receiving this packet records this initial information, verify the source node and extract some secrets that are only shared among the en route node and the source node.

$A(1)$ is $SID(1)||PC(1)||\delta_{R_1}(1)||...||\delta_{R_n}(1)$, where $||$ is concatenation of fields, $SID(1)$ is the source ID, $PC(1)$ is the count of the first packet, and $\delta_{R_j}(1)$ is the

[1] Data packets include all packet 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.

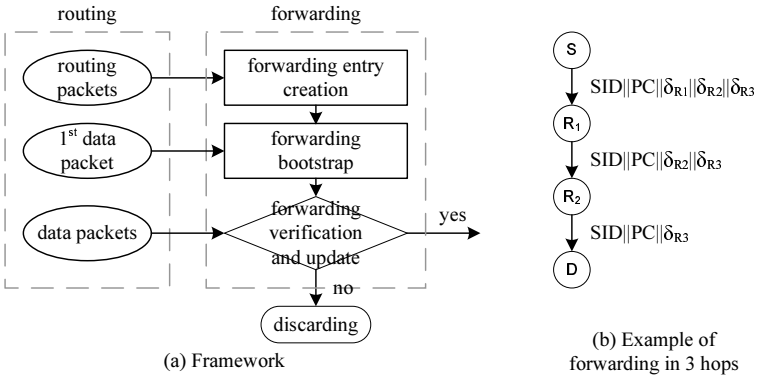


Fig. 1. Overview of SAF

authentication token for R_j. $\delta_{R_j}(1)$ is computed as $H(k_{S,R_j}, PC(1), \delta_{R_{j+1}})(1)$, where $H(*)$ is a hash function. The last token $\delta_{R_n}(1)$ is the packet signature so that the destination can verify the packet. Note that the length of a token is one byte. By computing a token for each en route node, the packet cannot be replayed to any other routing node in any other area in the ad hoc network, because only R_j can verify the token $\delta_{R_j}(1)$.

S appends $A(1)$ to the first data packet (thus the packet is called *bootstrap packet*), and sends it to all en route nodes toward D. Upon receiving the bootstrap packet, R_j verifies $\delta_{R_j}(1)$. If the verifications is successful, R_j stores $PC(1)$, remove $\delta_{R_j}(1)$ from $A(1)$, and forwards the bootstrap packet to the next hop R_{j+1}; otherwise, it discards the packet. After this procedure, each node has $SID(1)$, $PC(1)$ and the source route to identify the corresponding forwarding entry, and is ready for verification. An example of the forwarding procedure is shown in Figure 1(b).

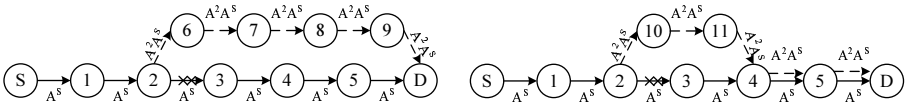
Update. For each new data packet $PKT(i)$, S composes a new authentication header $A(i)$ as $SID(i)||PC(i)||\delta_{R_1}(i)||...||\delta_{R_n}(i)$, where $PC(i)$ is one unit increment of $PC(i-1)$, i.e. $PC(i) \leftarrow PC(i-1)+1$. $\delta_{R_j}(i)$ is similarly computed as $H(k_{S,R_j}, PC(i), \delta_{R_{j+1}}(i))$. The whole packet is called *update packet* and sent into the route toward D.

Upon receiving $PKT(i)$, R_j first verifies $A(i)$. If the verification is successful, $PC(i)$ is greater than the last $PC(i')$ and $SID(i) = SID(1)$, R_j stores $PC(i)$, remove $\delta_{R_j}(i)$ from $A(i)$, and forwards the data packet to the next hop R_{j+1}. Otherwise, i.e. the verification fails or $PC(i) \leq PC(i')$, R_j discards the data packet. In case the bootstrap packet is lost, R_j will find that the verification is successful, but there is no $PC(1)$ or $SID(1)$ in the forwarding entry. Then, R_j treats $PKT(i)$ as the bootstrap packet and records $PC(i)$ and $SID(i)$ as $PC(1)$ and $SID(1)$ for this forwarding entry.

In summary, an en route node always keeps $SID(1)$, $PC(1)$, $PC(i)$ and the source route in the forwarding entry. These information helps an en route node to verify whether a packet is from the claimed source, and only the true source can use the route to forward data packets. Note that, since the forwarding module works in the network layer, it does not matter which application session in the source is using the route or whether TCP is used in the transport layer. For example, if TCP is used in a source's application, the source will treat a data packet retransmitted by TCP as two independent data packets in the forwarding module. Finally, the authentication header is added as an extension of the IP header of the packet, but is not authenticated as the IP header in IPSEC.

4 Forwarding in an Unreliable Ad Hoc Network

The simple version of forwarding module obviously overlooks the unreliability of the ad hoc network. Packet loss, route change and packet disorder might be fatal to the



(a) No overlap in the new segment (b) Some overlap in the new segment

Fig. 2. Forwarding in a new route

simple forwarding module. Hence, we extend its idea to handle these problems. In this section, we only consider the problems caused by the network itself. Some attacks can also result in similar problems or attackers may exploit these problems to attack the network. We will analyze these security issues in Section 5.

Packet Loss A packet could be lost due to communication error, hardware error, buffer overflow, etc. However, this will not affect the forwarding module. If the bootstrap packet is lost at R_j, en route nodes after R_j will treat the first received authentic data packet as the bootstrap packet for this forwarding entry. If an update packet is lost, the simple forwarding module will not be affected. Assume R_j successfully receives $PKT(i)$, but the next several packets are lost until R_j successfully receives $PKT(i')$. Since $PC(i') > PC(i)$ and R_j can still verify $A(i')$, $PKT(i')$ will be accepted. Hence, the loss of several update packets will not affect packet forwarding.

Route Change In an ad hoc network, a new route segment may be set up due to various reasons. For example, the routing protocol itself enables an en route node to overhear routing messages and discover shorter routes, or the route is broken due to link failures or the leaving of an en route node. If the new segment diverges from the previous one at the source node, the source node simply bootstrap a new forwarding procedure as in Section 3.3. However, if the new segment diverges from the previous one at an en route node, the situation turns to be very complicated and deserves more in-depth analysis.

Figure 2(a) depicts an example where the old route (solid lines) between S and D is broken at the link between nodes 2 and 3. If node 2 knows another route (dashed lines) that can reach D, the new segment diverges from the previous one at node 2. Note that nodes 3, 4 and 5 can still use the old segment to forward the packet, since the old segment is still valid at their positions and their buffered packets have valid authentication headers. Because S may not know the new segment immediately when the old route is broken, nodes 6 to 9 will not have any pairwise key with S before S starts a new forwarding procedure in the new route. In addition, some data packets may be already buffered in nodes 1 and 2 for forward, and S cannot modify the authentication headers in these packets. Hence, these buffered packets may be filtered even after S set a new route.

The idea to solve these problems is to let node 2 start another forwarding procedure in the new segment. Assume the old route is broken when node 2 tries to forward $PKT(\alpha)$ to node 3. Now, node 2 bootstraps a forwarding procedure in the new segment as illustrated in Figure 2(a). It basically appends its own authentication header A^2 to the authentication header A^S in each data packet. A^S is computed as described in Section 3.3 by the source; while A^2 is similarly computed as if node 2 was the source of the new segment, and thus SID in A^2 is node 2's ID.

Note that node 1 may not have any information about the new segment and do not have any information of the new 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[2]. Finally, we name the nodes that set up the forwarding procedure as *starter*, and SID in an authentication header is

[2] 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.

the starter ID instead of the source ID. In Figure 2, S is the starter in the old route, and node 2 is the starter in the new segment.

Packet Disorder As shown in the simple version, S increases PC for every update packet, and an enroute node only accepts an update packet with PC larger than the previous one. In this sense, PC represents the order of the packet delivery. However, if the order is mixed or reversed, the update packet with a smaller PC will be discarded in the simple SAF. We notice that the disorder can be caused by two reasons: either an attacker intentionally change the order of the packets, or a route is changed. We will discuss the first reason in Section 5.

The second reason can be illustrated in Figure 2(b), where the solid lines represent the old route, the dashed lines represent the new segment, and both 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 11 will reach node 4 before the old packets buffered in node 3 do. 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 as in the simple SAF.

To solve this problem, we make a small modification for the decision making. Based on the solution for route change, node 4 actually have obtained two PCs from $A^S(1)$ and $A^2(\alpha)$, which are $PC(1)$ and $PC(\alpha)$ (assuming the new segment is set up by node 2 when forwarding $PKT(\alpha)$). Obviously, $PC(i')$ in any packet buffered in node 3 satisfies $PC(1) \leq PC(i') < PC(\alpha)$; while $PC(i)$ in any packet going through node 11 satisfies $PC(\alpha) \leq PC(i)$. Hence, we let node 4 record the latest PC for each interval respectively, denoted as PC^S for $[PC(1), PC(\alpha))$ and PC^2 for $[PC(\alpha), \infty)$. When node 4 receives an update packet $PKT(i)$, it compares $PC(i)$ in the packet with either PC^S (if $PC(i)$ falls in $[PC(1), PC(\alpha))$) or PC^2 (if $PC(i)$ falls in $[PC(\alpha), \infty)$).

In summary, the decision making process is combined with the solution for route change as follows. First, each en route node stores different intervals of PC based on all bootstrap packets it receives. For example, in Figure 2(b), nodes 4, 5 and D have two intervals $[PC(1), PC(\alpha))$ and $[PC(\alpha), \infty)$; nodes 10 and 11 have one interval $[PC(\alpha), \infty)$; and nodes 1, 2 and 3 have one interval $[PC(1), \infty)$. Second, each en route node compares PC with the latest count in the corresponding interval. In this way, regular packets will not be discarded, if they are disordered due to route change.

5 Security Analysis

In this paper, we assume the attacker does not have the ability to break into the authentication tokens when the hash function is secure. Hence, no node can impersonate another node to send the data packet, and no node can verify a data packet which is not designated to it. When an attacker is "legally" injecting in the network as the source node, SAF easily exposes its ID, although we cannot prevent or stop such a "legal" injection. In the following security analysis, we do not consider the attacker or anyone of its coalition as a "legal" source. On the contrary, the attacker or its coalition could be a starter as node 2 in Figure 2, or just an en route node. We always use Figure 2 as the example for the following security analysis.

5.1 Packet injection

Property 1. If the attacker is an en route node, the probability that a forged packet can survive is negligible.

In order that an injected packet will not be filtered in the route, the attacker (as an en route node) should provide valid authentication tokens. The attacker needs to guess the tokens, since we assume the hash function is secure. Because the length of each authentication token is 1 byte, the probability that a forged packet can be accepted by the next hop is $\frac{1}{256}$. Accordingly, the probability that a forged packet can

go through m hops is only $(\frac{1}{256})^m$. Hence, SAF can effectively filter out extra junk data packets injected by malicious en route nodes with high probability.

Property 2. If the attacker claims to be a starter, it must use its own ID to authenticate data packets it need to forward or want to inject.

Assume node 2 in Figure 2 is the attacker claiming that the route is broken and it needs a new route to forward packets. By doing so, node 2 can inject extra packets into the new route. Node 2 must provide authentic A^2; otherwise, a data packet with a forged A^2 will be detected and discarded by node 2's next hops. But node 2 may forge A^S in extra update packets for injection. In Figure 2(a), these forged packets will not be filtered by nodes 6 to 9, since they do not verify U^S. However, D can detect them, because D can verify both A^2 and A^S. If the new segment is the one in Figure 2(b), the forged packet will be detected by node 4. Hence, the attacker can "legally" inject in the part of the new segment that does not overlap with the old route, but SAF exposes its ID to the destination and the nodes in the overlapped route segment.

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 it needs to forward. In the following, we analyze how the misuse may impact the regular packet forwarding. We find that misuse of SAF in general results in the drop of misused data packet, but does not affect other legitimate data packets.

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

The attacker can modify any field in the authentication headers. The modification will easily result in the failure of verification. 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, if the bootstrap packet or any update packet is dropped, SAF is not affected.

Property 4. If an attacker replays a packet, the packet will be discarded.

An attacker may replay previous update packets in order to inject extra packets in any area of the network. If the packet is replayed in the same route, SAF easily filters it, since a good en route node only accepts the update packet whose PC is greater than the PC in the last data packet. If the attacker replays a previous update packet but increases PC in the replayed packet, the packet will be discarded since its authentication token AUT can be verified based on the increased PC. If the packet is replayed to other node not in the route, the packet will be filtered, because it cannot be verified by any other node outside the route.

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

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

6 Performance Evaluation

In this section, we use simulations in NS2 [20] 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.

6.1 Simulation settings

Communication model. In the physical layer, the two-ray ground reflection model is used to model the signal propagation in the simulations. We choose the widely used IEEE 802.11 as the MAC and PHY protocols for the communications among mobile nodes. The CSMA and DCF functions are used to avoid transmission collision among nearby nodes. Each node has a sensing range of 550 meters and 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.

Network and traffic. 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 $2m/s$, $5m/s$ 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 load of all connections and the maximum speed of all nodes are the same. For each scenario, we randomly generate 10 cases for simulation.

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

6.2 Impacts of SAF on Regular Traffic

Overhead of SAF SAF appends an authentication header of several bytes to every data packet, which include the starter ID, the packet count, and authentication tokens. The size of the authentication token changes 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[3]. Figure 3 shows the average overhead vs. the total hops of a path. Each sub figure shows the overhead at different maximum speeds, and each line in a sub figure represents the overhead at different loads.

As illustrated, the authentication header is larger when the path is longer. This could happen when the destination is far away from the source, or the network is unreliable that 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 only 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

[3] Here, 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.

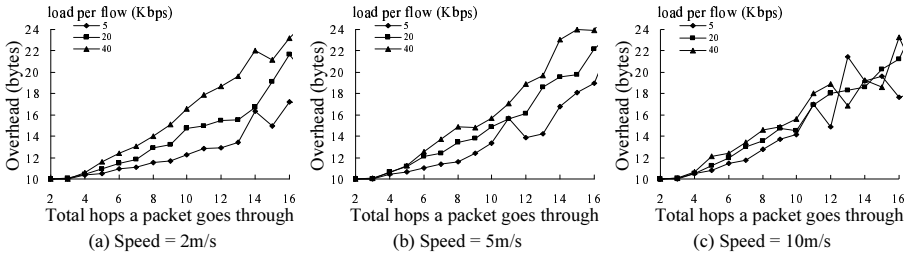


Fig. 3. Communication overhead per hop

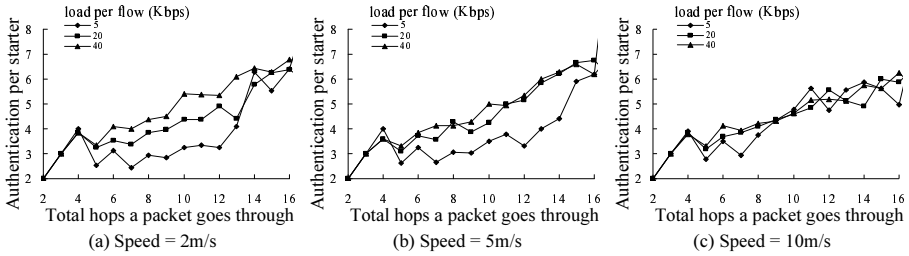
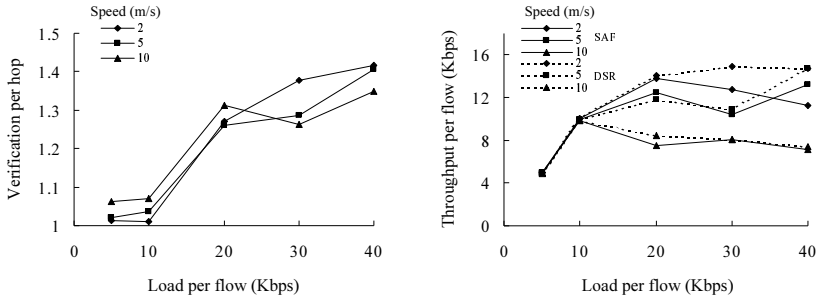


Fig. 4. Number of authentication tokens a starter needs to compute

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 add more than 1 byte to the average overhead. Furthermore, when the speed is low, the difference of slopes under various loads is obvious. While the speed is high, such different is diminished. 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.

Computation of SAF The starter needs to compute authentication headers for data packets and each en route node needs to verify packet sources. The computation demand for starters, measured as the number of authentication tokens a starter needs to compute, is depicted in Figure 4. Differing from the overhead, the computation for authentication does not increase as much as overhead when the path is longer. As we trace into 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 is longer, which can be inferred from the average overhead of the path. Similar to overhead, unreliability (higher load and speed) increases the computation for starters (although slightly). In the worst case (40Kbps and 10m/s), a starter needs to computer around 0.3 authentication token in average for each hop in the path.

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 5(a). 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 low (5Kbps to 10Kbps), a little more than 1 verification is needed in each hop for each data packet. In the cases of light loads, it is clear more verification is needed when the speed is higher. 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



(a) Number of authentication tokens a hop needs to verify

(b) Throughput comparison

Fig. 5. Verification Cost and Throughput

segments. Hence, 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 another word, 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.

Throughput of SAF Finally, we use Figure 5(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 becomes the factor to reduce the throughput. However, the impact is slight or statistically insignificant. For example, when the speed is $10m/s$, the throughput of SAF is very close to that of DSR. Hence, the design of SAF almost does not interfere with the regular data packet forwarding, although some overheads are added in the network.

Figure 5(b) also implies that the packet injection attacks could seriously disrupt the legitimate traffic. According to the dashed lines (where attackers can inject because only DSR is applied), when the traffic load is $40Kbps$, the throughput is only $15Kbps$, which means more than 62% of packets are dropped. Assume some flows are injected traffic in the simulation, the network is disrupted by attackers without the protection of SAF. Figure 5(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. Note that the network may be disrupted by the legitimate loads, since SAF does not set any rate limit on legitimate traffic. Hence, the solid lines, which stand for the throughput of legitimate traffic, indicates that SAF not only protects the network, but also does not interfere with the normal network traffic.

7 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. This protocol is designed to fit in the unreliable environment of ad hoc networks. The protocol can either immediately filter 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 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, and this protocol can be integrated into other hop-by-hop source authentication protocols to improve their performance in an unreliable ad hoc network.

References

1. Hu, Y.C., Perrig, A., Johnson, D.B.: Ariadne: A secure on-demand routing protocol for ad hoc networks.pdf. In: ACM MobiCom. (2002) 12–23
2. Hu, Y.C., Perrig, A., Johnson, D.B.: Sead: Secure efficient distance vector routing for mobile wireless ad hoc networks. Ad Hoc Networks **1** (2003) 175–192
3. Zapata, M.G., Asokan, N.: Securing ad hoc routing protocols. In: ACM workshop on Wireless Security, Atlanta, GA, USA, ACM Press New York, NY, USA (2002) 1–10
4. Ye, F., Luo, H., Lu, S., Zhang, L.: Statistical en-route detection and filtering of injected false data in sensor networks. In: IEEE Infocom. (2004)
5. Zhu, S., Setia, S., Jajodia, S., Ning, P.: 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. Sanzgiri, K., Dahill, B., Levine, B.N., Shields, C., Belding-Royer, E.M.: A secure routing protocol for ad hoc networks. In: IEEE ICNP. (2002) 78–89
7. Yi, S., Naldurg, P., Kravets, R.: Security-aware ad hoc routing for wireless networks. In: ACM MobiHoc. (2001) 299–302
8. Papadimitratos, P., Haas, Z.: Secure routing for mobile ad hoc networks. In: SCS Communication Networks and Distributed Systems Modeling and Simulation Conference, San Antonio, TX (2002)
9. Park, J.M., Chong, E.K.P., Siegel, H.J.: Efficient multicast packet authentication using signature amortization. In: IEEE Symposium on Security and Privacy. (2002) 227
10. Wong, C.K., Lam, S.S.: Digital signatures for flows and multicasts. IEEE/ACM Transactions on Networking **7** (1999) 502–513
11. Golle, P., Modadugu, N.: Authenticating streamed data in the presence of random packet loss. In: NDSS. (2001) 13–22
12. Park, J.M., Chong, E.K.P., Siegel, H.J.: Efficient multicast stream authentication using erasure codes. ACM Transactions on Information and System Security **6** (2003) 258–285
13. Perrig, A., Canetti, R., Tygar, J., Song, D.: Efficient authentication and signing of multicast streams over lossy channels. In: IEEE Symposium on Security and Privacy, Berkeley, CA (2000) 56–73
14. IEEE: Wireless lan medium access control (mac) and physical (phy) layer specification (1999)
15. Johnson, D., Maltz, D., Hu, Y.C., Jetcheva, J.: The dynamic source routing protocol for mobile ad hoc networks (dsr), ietf internet draft, draft-ietf-manet-dsr-09.txt (2002)
16. Perkins, C., Royer, E., Das, S.R.: Ad hoc on-demand distance vector (aodv) routing, ietf internet draft, draft-ietf-manet-aodv-11.txt (2002)
17. Du, W., Deng, J., Han, Y.S., Varshney, P.K.: A pairwise key pre-distribution scheme for wireless sensor networks. In: ACM CCS. (2003) 42–51
18. Liu, D., Ning, P.: Establishing pairwise keys in distributed sensor networks. In: ACM CCS. (2003) 52–61
19. Zhu, S., Xu, S., Setia, S., Jajodia, S.: Establishing pairwise keys for secure communication in ad hoc networks: a probabilistic approach. In: IEEE ICNP. (2003) 326–335
20. NS2: The network simulator, http://www.isi.edu/nsnam/ns/ (2004)