Slander Resistant Attacker Isolation in Ad Hoc Networks

Qijun Gu, Chao-Hsien Chu, Peng Liu, Sencun Zhu
Cyber Security Lab
School of Information Sciences and Technology
Pennsylvania State University
University Park, PA 16802

Abstract

This paper focuses on how to isolate attackers that inject packets to cause denial-of-service (DoS) effects in ad hoc networks. Our security analysis shows that current hop-by-hop source authentication protocols only partially achieve the defense goals, although they allow forwarding nodes to effectively identify and discard injected or modified packets. The other important defense goal, which has not been achieved yet, is to isolate the attackers so that they cannot inject in the future. Current authentication protocols provide evidence of injection attacks, since injected packets will incur verification failures. Nevertheless, the evidence may be exploited by attackers to deceive defenders. We find that a non-injection attacker can slander any good forwarding node in a route by modifying the authentication information carried in the packets. In order to correctly isolate suspicious nodes, we propose a new authentication approach. The approach not only preserve the function to filter junk packets as in current authentication approaches, but also help to isolate the attackers with a high probability. This approach ensures that defenders can focus on investigating only two nodes to find out the real attacker once failed verifications are detected.

1 Introduction

Since the network resources are typically very limited, ad hoc networks are vulnerable to various DoS attacks. In this paper, we are mainly concerned with the defense against the packet injection attack because the attack is easy to enforce but hard to defend. In an ad hoc network, a node generally forwards packets based on the destination address. Hence, an attacker can easily spoof legitimate nodes to inject packets into good routes without being traced back. Once a junk packet is injected in a route, it can affect the forwarding nodes until the packet is dropped. Since wireless channels are a shared media, injecting junk packets into legitimate routes may not only cause serious network congestion at the target side, but also lead to severe wireless channel contention along the routes.

To defend against packet injection attacks, source authentication is often used for two defense purposes. First, authentication helps forwarding nodes to identify and discard injected packets. An authentication protocol requires a source to authenticate each packet before sending the packet out. When a forwarding node receives a packet, it will verify the packet first. Only if the verification passes, the node will forward the packet. Because authentication information is cryptographically computed based on the credentials only known by the source, it is not easy for the other nodes to forge a packet with correct authentication information. Hence, most forged packets will fail the verification and be discarded, and thus the network is protected. Second, authentication provides evidence of injection attacks. Failed verifications are the trail of attackers. Because only the source can authenticate the packets, a failed verification indicates that a malicious forwarding node in the route is modifying or forging authentication information. Following the trail, defenders can identify the attackers from the network. Only after isolating suspicious nodes, the network can be protected.

Regarding the two defense purposes, existing authentication approaches are not sufficient. On the one hand, most authentication approaches [5, 16, 21, 22, 23, 25] are only designed to protect end-to-end data integrity. When an attacker modifies the content of a packet, receivers can quickly detect the modification by verifying the authentication information carried in the packet. Nevertheless, if the attacker replays authenticated packets during the lifetime of the packets, forwarding nodes will not discard the packets because the packets are not changed. In addition, because these protocols are mainly designed for end-to-end authentication, forwarding nodes are not able to verify and filter out injected packets upon receiving the packets. Thus, junk packets can go through the route until the end nodes discard them. Longer a junk packet stay in a route, more it affects legitimate traffic. Hence, these protocols cannot address the injection attacks. On the other hand, a few hop-by-hop authentication protocols [26, 29] are designed to enable forwarding nodes to immediately filter out injected packets. However, they cannot help defenders to identify the attackers. As we will discuss in detail later, when the verification of a packet fails, defenders cannot tell whether the packet is injected by an attacker or the packet's authentication information is modified by a non-injector. In some circumstance, an attacker can even exploit the failed verification to slander a good node. Hence, it raises the question for defenders how to decide which node is injecting with hop-by-hop source authentication protocols.

In this paper, we present a new hop-by-hop source authentication protocol. The contribution of this protocol lies in the new approach to authenticate packets. First, the authentication is based on Chinese Remainder Theorem that allows the source node to integrate separate pieces of authentication information into one. In this approach, each packet still carries an authentication header, which can be verified by each forwarding node in the route. If a malicious forwarding node modifies or forges a packet, its next hop can identify and discard the packet. Hence, the approach preserves the function to filter out junk packets as in current authentication approaches. Second, the authentication header cannot be forged or modified by any of the forwarding nodes. If a malicious node modifies an authentication header, its next hop can also detect the modification. Hence, a failed verification cannot be manipulated by any forwarding node to slander another node. This help defenders to isolate the attackers. As we show in the paper, this approach ensures that defenders can focus on investigating only two nodes to find out the real attacker once failed verifications are detected.

The rest of the paper is organized as follows. Section 2 introduces the background on the attack and the defense, and the motivation of our study. In Section 3, we propose a CRT-based approach to compute authentication headers. The security of the approach is discussed in Section 4. We evaluate the performance of our schemes in Section 5. Section 6 describes the related work. Finally we conclude the paper in Section 7.

2 Background and Motivation

In this section, we present the injection attack and the defense. We show that although current source authentication approaches can verify and discard junk packets, they cannot help defenders to identify which node really injects or modifies packets. This problem motivates us to develop a new authentication scheme.

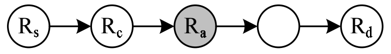


Figure 1: Injection attacks in a route, in which R_s is the source and R_d is the destination. An attacker R_a stays in the route and injects junk packets with the spoofed address of R_s. Because the route is used to forward packets from R_s to R_d, junk packets will be forwarded. The destination can trace the packet back along the route if it detects junk packets. Unfortunately, the trace goes back to the source R_s instead of the attacker R_a in the middle.

2.1 Injection Attacks

We mainly study injection attacks in unicast communication. A node launches the packet injection attacks because it has been compromised or it intentionally does. 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 it is risky for an attacker to misbehave in its own name in ad hoc networks, where the routing protocols [10, 18] can easily help defenders to trace to the packet source.

To inject packets with spoofed source addresses, the attackers can take the following two steps. First, the attackers need to find valid routes toward the targets in the network. The attackers may follow the routing protocols to discover good routes by themselves. The attackers may also eavesdrop their nearby traffic, from which they can identify whether their neighbor nodes have valid routes. The attackers can also check their own routing entries and may find that they are already in valid routes. Then, the attackers inject junk packets into the valid routes as shown in Figure 1. In this step, the attackers spoof the source addresses in the junk packets, but they can claim that they are forwarding these packets from the claimed sources.

Multiple attackers may collude in an attack to hide themselves. In some intrusion detection approaches [9, 15, 28], a forwarding node is required to monitor its neighboring nodes for misbehavior and restrict malicious packet transmission. For example, in Figure 1, R_c may be required to monitor the transmission of R_a. However, if R_c is a collaborator of R_a, R_a can inject without being monitored. Colluding attackers may also want to break defenses or exploit flaws in defenses to cause other security problems. For example, they can share the credential assigned by defenders. If the deployed

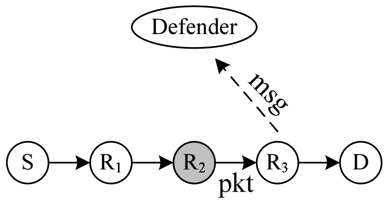


Figure 2: Defense architecture in an ad hoc network, in which the source S is sending packets to the destination D. The source must authenticate each packet to be sent, and a forwarding node must verify each packet to be forwarded. In the route, an attacking node is injecting packets by spoofing S, or slandering a good node by modifying packets.

defense approach is only resistant to m colluders [1], $m + 1$ attackers can break the defense system.

2.2 Defense Framework

Figure 2 depicts how a network is protected by a defense system. In the network, we assign a trusted node as a defender, which can be the access point of the network or the service node in the network. We assume the defender is well protected and trusted. The defender is responsible to receive reports from forwarding nodes and alert other forwarding nodes to avoid suspects in their routes.

The defense works in two phases. In the first phase, all nodes forward packets according to an authentication protocol in the network. Because an attacker may inject junk packets in a route, a forwarding node must verify each packet pkt in order to filter junk packets. The second phase happens when a forwarding node identifies junk packets. The forwarding node reports msg which includes the information of the junk packets to the defender. Based on the reports, the defender tries to identify and isolate potential attackers. The defender can broadcast the identities of potential attackers in the network [14] so that the other nodes can avoid to include them as forwarding nodes into their routes.

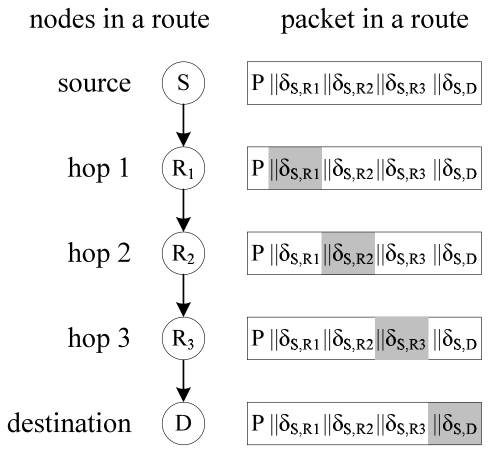


Figure 3: Forwarding in a route, which consists of five nodes, include the source and the destination. When the source sends a packet, it computes four authentication tokens for the three forwarding nodes and the destination, and || stands for the concatenation of the packet and the tokens.

2.2.1 Source Authentication

As previously discussed, hop-by-hop source authentication [26, 29] is the technique to filter injected packets. The protocol takes three steps. First, the source and the destination establishes a route. In this step, the source node obtains the IDs of the forwarding nodes. This information is readily available in the ROUTE REPLY packet of the routing protocol DSR [10]. We assume that all routes are secured by secure routing protocols [7, 8, 27], so that a node in a route shows its true identity.

Then, the source node sets up symmetric keys with the en route nodes. The simplest way to set up symmetric key is to pre-load keys into nodes, although the memory requirement is not tolerable when the network is large. The literature provides many novel key management schemes with better performance. For example, in random key schemes [4, 3, 13, 30], any two nodes can establish a symmetric key with a sufficiently high probability. At the same time, by encrypting with the symmetric keys, the source can securely assign certain credentials to each forwarding node. Finally, the source node computes authentication tokens as depicted in Figure 3. For each packet P, the source computes and appends four tokens: δ_{S,R_1}, δ_{S,R_2}, δ_{S,R_3} and $\delta_{S,D}$. In [26], the computation of a token is, for example, $\delta_{S,R_1} = H_{k_{S,R_1}}(P)$, where k_{S,R_1} is a symmetric key only known to S and R_1 and $H_k(*)$ is a keyed hash function.

Once a forwarding node receives a packet, it will first verify the corresponding token (the gray token in Figure 3). For example, R_1 verifies δ_{S,R_1}, R_2 verifies δ_{S,R_2}, R_3 verifies δ_{S,R_3}, and D verifies $\delta_{S,D}$. The forwarding nodes will forward the packet only if the verification passes. Otherwise, the packet will be discarded. Because each token is computed based on a secret (the symmetric key in this example) shared between the source and the corresponding node, it is hard for a malicious forwarding nodes to forge a correct token. Hence, the hop-by-hop source authentication ensures that a packet modified or forged by a forwarding node will be discarded by the node's next hop.

2.2.2 Attacker Isolation

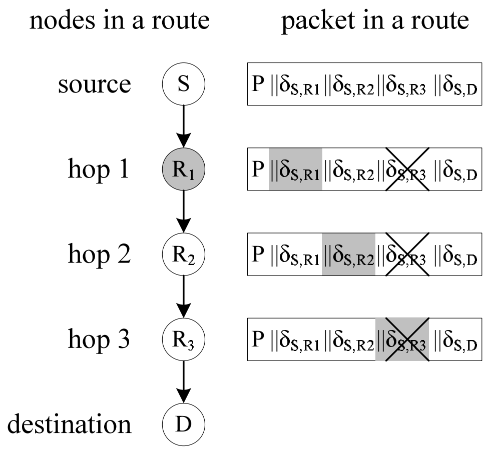
Filtering junk packets only partially achieves defense objectives, since the defender cannot identify and isolate attackers at this stage. Because failed verifications indicate misbehavior in the route, they can be used as evidence to infer the real attackers. Generally, the following isolation rule is used to identify and isolate attackers in a route, once a forwarding node reports failed verifications to the defender.

Isolation rule: Assume node R_j is a reporter, and node R_{j-1} is R_j's previous hop. If R_j reports a failed verification, the defender isolates R_{j-1} and R_j.

Obviously, when an attacker forges a packet, its next hop can detect the forged packet. Hence, the defender knows the previous node is the attacker. Furthermore, because it is possible that the reporter is hoaxing, the defender should isolate both the reporter and its previous hop. In Section 3.3, we show that the isolation only restricts the malicious nodes from forwarding packets. An isolated node can still send or receive packets as the source or the destination.

2.3 Misuse Attacks and Design Motivation

The isolation rule can be misused by non-injection attackers to slander a good node, because attackers can modify the tokens in a packet to cause the verification failures without being detected. An example is shown in Figure 4, where the attacker R_1 only tries to misuse the isolation rule to slander a good forwarding node. In this example, R_1 modifies δ_{S,R_3}, but keeps all other tokens in the packet intact. When R_2 receives the packet, R_2 will successfully



nodes in a route
packet in a route
source
S
P ||δS,R1||δS,R2||δS,R3 ||δS,D
hop 1
R1
P ||δS,R1||δS,R2||δS,R3 ||δS,D
hop 2
R2
P ||δS,R1||δS,R2||δS,R3 ||δS,D
hop 3
R3
P ||δS,R1||δS,R2||δS,R3 ||δS,D
destination
D

Figure 4: Slander a good node. The route consists of five nodes, including the source and the destination. R_1 is the attacker that modifies δ_{S,R_3} to slander R_2.

verify the packet because δ_{S,R_2} and P were not modified. Then, R_2 forwards the packet to R_3. Because δ_{S,R_3} was modified, the verification in R_3 will fail. By following rule 1, R_3 will report R_2 as the attacker[1]. In this way, R_1 slanders R_2 without being detected. R_2 may be isolated due to the failed verification.

The example shows that although hop-by-hop source authentication can filter junk packets, it cannot help to identify the real attacker. On the one hand, only when the report is caused by an injection attack, the defender can isolate attackers according to the isolation rule. On the other hand, if the report is caused by a misuse attack, any node from the source to the reporting node in the route could be an attacker that tries to slander a good node. In fact, when the defender receives the report from a forwarding node, the defender cannot decide whether the report is caused by injection attacks or by misuse attacks. Consequently, the defender cannot decide which nodes need to be isolated according to the failed verifications.

The problem lies in the design of the tokens. In the basic approach, they are separately computed for each forwarding nodes. Hence, our idea is to

[1]It is impractical to ask the forwarding nodes in the route to report whether their verifications pass or not, because the forwarding nodes may not memorize the packets they have forwarded

integrate the tokens into one so that it is very hard for a forwarding node to modify it. With the newly designed authentication token, the number of suspects is reduced, and the defender can focus on investigating and isolating less nodes.

3 Defense Design

In this section, we present a new design of the authentication head. We first describe the format of a packet and a report message. Then, we discuss in detail how to compute and verify an authentication header. Finally, we discuss the isolation procedure.

3.1 Packet and Message Design

In our proposed defense system, we use an integrated authentication token to replace the separated tokens that are used in the basic authentication approach presented in Section 2.2.1. As shown in Figure 2, the packet pkt forwarded in the link between R_{j-1} to R_j consists of three fields:

$$pkt = P||N||\alpha$$

P is the original data packet. N is a sequence number of the packet, which is increased by the source for each new packet in the route. α is the new designed authentication token, which is a single token for all forwarding nodes. N is used to prevent replaying attacks. When the forwarding node R_j receives a packet, it verifies whether N in the received packet is larger than N in the previous packet. Besides comparing N, R_j also needs to verify α. α is computed in a way that all forwarding nodes can verify it but no one can easily forge it (see details in Section 3.2). If the current N is larger and α passes the verification, R_j forwards the packet to its next hop.

When an attacker R_{j-1} replays packets or forges packets, its next hop R_j will detect the injection behavior. Then, R_j reports the message msg to the defender. msg includes the replayed or forged packet (P), the authentication information ($N||\alpha$), and the copy of the current route R. $H(*)$ is the hash of the three information. msg is encrypted and authenticated with the symmetric key K_{R_j} that is only shared between the defender and R_j. K_{R_j} is assigned to R_j by the defender, when R_j joins the network. Obviously, any other node cannot forge a msg in the name of R_j. $P||N||\alpha$ is the evidence of injection, and R is used for the defender to further investigate the suspects.

$$msg = \{P||N||\alpha||R||H(P||N||\alpha||R)\}_{K_{R_j}}$$

3.2 CRT-based Authentication

Because the basic approach to compute the authentication information is not suitable for the defender to isolate attackers, we propose the following scheme to design a new authentication token. In this scheme, first, a token is generated randomly in the point of view of the forwarding nodes, and no forwarding node can infer other tokens from any received token; second, a token contains some information shared between the source node and the forwarding nodes so that it can be partially verified by these forwarding nodes; and third, if a token is forged by an inside attacker, the bogus token can be detected with a high probability by the other forwarding nodes.

This approach is based on Chinese Remainder Theorem (CRT) [2, 24, 31]. (An overview of CRT is given in Appendix A.) For simplicity, we only use prime numbers instead of relative prime numbers to describe the key generation approach used in CRT. This CRT-based approach will be more secure if relative prime numbers are used. To present the design, we assume the route has n forwarding nodes, $R_1, ..., R_n$. The source node S and each forwarding node R_j share a symmetric key k_{S,R_j} as described in Section 2.2.1. The source node also uses k_{S,R_j} to secretly assign a prime number m_j to each forwarding node R_j.

3.2.1 Authentication Token

The source node computes a random token α for the data packet P in four steps. First, it computes n random numbers for all forwarding nodes as in the basic hop-by-hop authentication protocol.

$$\delta_j = \delta_{S,R_j} = H_{k_{S,R_j}}(P||N)$$

Then, it computes the remainder for each forwarding node R_j.

$$r_j = \delta_j \ mod \ m_j$$

.

Third, it computes the authentication token

$$\alpha = \sum_{j=1}^{n} c_j r_j \ mod \ M$$

Where c_j is a coefficient and $M = \prod_{j=1}^{n} m_j$. c_j is pre-computed as $c_j = M_j \times (M_j^{-1} \; mod \; m_j)$, where $M_j = \frac{M}{m_j}$.

Finally, it uses a random number Q to randomly permute α.

$$\alpha = \alpha + M \; with \; 50\% \; probability \; , \; if \; \alpha < Q$$

Q is a positive number randomly picked by the source. In order to limit the length of α, Q is in the range of $(0, 2^{l(M)} - M)$, so that α has no more than $l(M)$ bits.

Obviously, the authentication token satisfies the congruence equations

$$\alpha \equiv r_j \equiv \delta_j (mod \; m_j), \; for \; 1 \leq j \leq n \tag{1}$$

3.2.2 Verification

Upon receiving the data packet pkt, the forwarding node R_j first obtains P and N, and then computes $\delta'_j = H_{k_{S,R_j}}(P||N)$. R_j checks whether α is congruent to δ'_j modulo m_j (i.e. $\alpha \equiv \delta'_j(mod \; m_j)$) according to Eq.(1). At the same time, it checks whether the current N is larger than the previous N. If both verification results are correct, R_j accepts pkt and forwards it to the next hop. Otherwise, R_j discards pkt.

In summary, the CRT-based authentication token integrates the individual authentication tokens that are used in the basic authentication protocol. All forwarding nodes verify the new token in a received packet, in stead of verifying individual tokens. Because a forwarding node does not know the prime number of any other node, the token cannot be modified or forged. Therefore, when a forwarding node detects a failed verification, it is highly possible that its previous hop forged or modified the token (see analysis in Section 4).

3.3 Isolation

The CRT-based authentication token ensures that a forged or modified packet will be detected in one hop. Hence, the isolation rule in Section 2.2.2 cannot be misused by a forwarding node to slander a good node without being detected.

Note that the isolation does not mean to block any traffic to or from the isolated nodes. The defense emphasis is on preventing an attacker from injecting packets in the name of a good node in a route. When the isolated nodes are sending or receiving packets as the source or the destination with their own identities, there is no need to block them. Therefore, the isolation only forbids the isolated nodes to forward in a route so that they do not have any chance to inject without being detected. At the same time, the isolation does not interfere with legitimate activities of the suspects in their own names.

The isolation is implemented in two steps. First, the defender broadcasts the IDs of the isolated nodes. The security of the broadcast can be ensured by some secure broadcast protocols [2, 19]. Second, when a good node receives the broadcast, it excludes the isolated nodes from its cached routes, where the isolated nodes are the forwarding nodes. Nevertheless, if the isolated are the source or the destination in the routes, the routes can be kept. In this way, the isolation immediately restricts the isolated nodes to inject junk packets or slander other nodes in the network.

Because an isolated node is not restricted to send or receive packets as a source or a destination, a report does not affect the reporter or the reportee to access their services. A report only makes the two nodes out of the forwarding procedure in the network. Compared with the tokens that are used in the existing authentication protocol, a CRT-based token will allow the defender to only isolate two nodes. This paper does not address how to further investigate the isolated nodes to find out whether the reporter is hoaxing. This puzzle solving problem will be studied in our future work.

4 Security Analysis

An attacker needs to forge α to inject junk packets or modify α to slander a good node. With the CRT-based key generation, α needs to satisfy the congruence condition in Eq.(1) so that the forwarding nodes will not discard the forged packets. In the following, we first analyze how possible a forged token will not be detected in a forwarding node. Then, we analyze how difficult an attacker slander a good node. For analysis, we assume R_j is an attacker and R_{j+1} is its next hop.

4.1 Injection Analysis

To inject junk packets, an attacker needs to provide valid tokens. Either the attacker can try to break the received tokens to obtain the credentials for computing a token, or the attacker can guess a token. We show in the following that neither way is feasible.

Property 1 *It is computationally infeasible for an attacker to break tokens and obtain symmetric keys for other en route nodes.*

To inject junk packets, the attacker needs to provide valid tokens for the junk packets, which require the attacker to know the symmetric keys and the prime numbers for other en route nodes. For simplicity, we assume that the attackers know all prime numbers. Then, the difficulty to break a hash function is determined by the length of a symmetric key which is used in the computation of the keyed hash function $H_k(*)$. Assume that the length of a key is h-bit. Given a packet and its hash, $O(2^{h-1})$ computations are required to break the hash and obtain the key. Without knowing the prime numbers, more computation is required to break a token. Hence, it is computationally infeasible for an attacker to break a token, as long as the hash function is collision free and the key is long.

Property 2 *The probability that a forged packet will not be filtered is negligible.*

Although it is infeasible for an attacker to break into tokens and obtain symmetric keys for other en route nodes, the attacker can guess tokens. Hence, whether an attacker can forge a correct token is determined by whether the guessed token can satisfy the congruence condition in Eq.1. Assume that the prime number of the attacker's next hop is m_{j+1}. The attacker has a 1 in m_{j+1} chance to forge a token that satisfies the congruence condition. For example, in this study, we use prime numbers with at lest 8 bits. The probability that a forged packet can be accepted by the next hop is less than $\frac{1}{128}$. Hence, a forged packet is very possible to be detected and discarded in one hop.

Property 3 *The probability that an attacker will not be isolated is negligible.*

An attacker will not be isolated only if the injected junk packets can go through its next hop without being detected. However, because the probability for an attacker to successfully guess a token for the next hop is very small, the next hop can easily detect the junk packets and report to the defender. Hence, the attacker will be isolated.

4.2 Slander Analysis

It is important for the attacker R_j to know which prime numbers are assigned to other forwarding nodes in order to slander a good node. Assume that R_j knows the prime number m_{j+1} assigned to R_{j+1} by chance. R_j can first figure out r_{j+1} from α, i.e. $r_{j+1} \equiv \alpha(mod\ m_{j+1})$. Then, R_j can forge another token α', which satisfies $\alpha' \equiv r_{j+1}(mod\ m_{j+1})$, based on the CRT approach. When R_{j+1} receives the packet, R_{j+1} will forward the packet to R_{j+2} because the verification of α' will not fail. However, α' will not pass the verification in R_{j+2}, and thus R_{j+2} will report to the defender to isolate R_{j+1}. Fortunately, it is hard for R_j to know m_{j+1} as follows.

Property 4 *In the CRT-based token generation approach, attackers (a) do not know which prime numbers the source node assigns to the forwarding nodes and (b) do not know M.*

R_j cannot easily know the prime numbers assigned to other forwarding nodes. First, a prime number is secretly assigned to a forwarding node by the source. Because R_j cannot obtain the key as previously analyzed, R_j cannot obtain the prime number. Second, the permutation in token computation prevents R_j from inferring M from any token. If R_j knows M, it can derive all assigned prime numbers by factorizing M. Because the source node will not disclose M, an attacker has to guess M from all received tokens. With the permutation, a token is in the range $[0, M+Q)$. Because Q is only known to the source node, the attacker cannot know M.

Property 5 *The probability that an attacker correctly guesses k prime numbers assigned to its downstream forwarding nodes is negligible.*

Still assume that a route has n hops, and R_j is an attacker. Assume that R_j knows a minimum set of n' prime numbers from which the source node selects a subset of n prime numbers to generate its random keys. For example, the attacker knows that the source node selects 10 prime numbers (i.e., $n = 10$) for a 10-hop route and each number has 8 to 10 bits. Since there are 142 prime numbers having 8 to 10 bits, $n' = 142$. Note that an attacker does not have to guess which prime number is assigned to which forwarding node. For example, suppose R_j has correctly guessed two prime numbers m_{j+1} and m_{j+2}, but assign m_{j+1} and m_{j+2} to R_{j+2} and R_{j+1} respectively. When R_j forges α', it only has to make sure that α' satisfies two congruence equations: $\alpha' \equiv r_{j+2}(i)(mod\ m_{j+2})$ and $\alpha' \equiv r_{j+1}(mod\ m_{j+1})$. When R_{j+1} receives α', the second equation makes R_{j+1} accept α'; while the first equation makes R_{j+2} accept α'.

Furthermore, in order to guess k prime numbers, an attacker can guess more than k numbers without exactly knowing the assigned k numbers. For example, we assume that R_j guesses 2 prime numbers for R_{j+1}, which include an unassigned number m^*_{j+1} and the number m_{j+1} assigned to R_{j+1}. Then R_j forges α' that satisfies two congruence equations: $\alpha' \equiv r_{j+1}(mod\ m_{j+1})$ and $\alpha' \equiv r^*_{j+1}(mod\ m^*_{j+1})$. When R_{j+1} receives α', the first equation makes R_{j+1} accept α'.

Let $P(k)$ be the probability that R_j correctly guesses the prime numbers assigned to $R_{j+1}, ..., R_{j+k}$. Assume that the attacker guesses m keys, $m \geq k$. First, there are $C^m_{n'-1}$ possible ways that the attacker selects m prime numbers from n' prime numbers. Second, if the m prime numbers include the k prime numbers, there are $C^{m-k}_{n'-1-k}$ possible sets of such m prime numbers. Hence,

$$P(k) = \frac{C^{m-k}_{n'-1-k}}{C^m_{n'-1}} = \frac{(n'-1-k)!m!}{(n'-1)!(m-k)!} = \prod_{q=1}^{k} \frac{m+1-q}{n'-q} \tag{2}$$

Obviously, more prime numbers the attacker guesses, more successful the attack could be, i.e. larger m is better for the attacker. In a n-hop route, the attacker can guess n prime numbers to compute a key as the source node. Thus, the worst case is

$$P(k) = \frac{C^{n-k}_{n'-1-k}}{C^n_{n'-1}} = \prod_{q=1}^{k} \frac{n+1-q}{n'-q} \tag{3}$$

First, $P(k) = 1$ if and only if the attacker exactly knows which prime numbers are used by the source node, i.e. $n' = n + 1$. If the attacker cannot get the exact subset, the probability is in the range

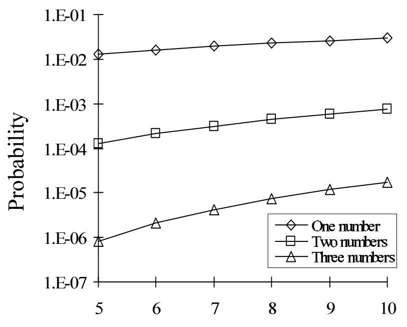
$$(\frac{n+1-k}{n'-k})^k < P(k) < (\frac{n}{n'-1})^k$$

Hence, $P(k)$ is decreased exponentially to the number of prime numbers an attacker wants to guess. As shown in Figure 5(a), given $n' = 309$ and a route with less than 10 hops, there are less than 3.3% chance that an attacker can correctly guess the prime number assigned to the next node, and less than 0.0025% chance to correctly guess the prime numbers assigned to the next 3 forwarding nodes. In another word, if the source node wants to ensure that an attacker has less than 3.3% chance to correctly guess the prime number of the next forwarding node, i.e. $P(1) = (\frac{n}{n'-1}) \leq 0.033$, it should use at least n' prime numbers as shown in Table 1.

Property 6 *Colluding attackers have slightly better chance to correctly guess the prime numbers.*

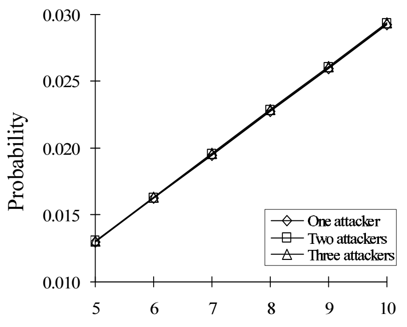
Table 1: Smallest n' to ensure less than 3.3% chance for an attacker to correctly guess one prime number assigned to the next forwarding node

hop number of a route, n	5	6	7	8	9	10
number of prime numbers, n'	153	183	214	244	274	305



Routes with different hop numbers

(a) Probability for one attacker to guess multiple prime numbers



Routes with different hop numbers

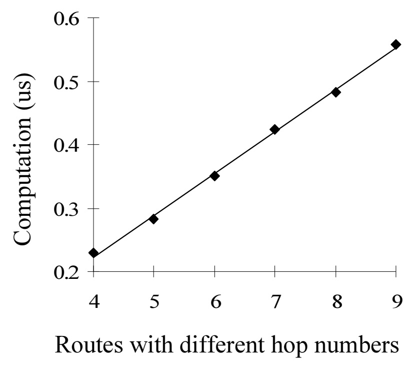
(b) Probability for colluding attacker to guess one prime number

Figure 5: The source node selects prime numbers from $n' = 309$ numbers

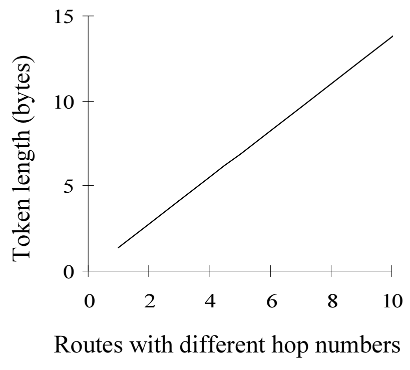
Assume that c colluding attackers in a route ($c \leq n - 1$) share their prime numbers. Although the source node uses n' prime numbers, the minimum set of prime numbers attackers know includes only $n' - c$ prime numbers, because colluding attackers can know all the c prime numbers assigned to them. An attacker can still guess n prime numbers. Thus, the probability they can correctly guess k prime numbers is

$$P_c(k) = \frac{C^{n-k}_{n'-c-k}}{C^n_{n'-c}} = \prod_{q=1}^{k} \frac{n+1-q}{n'-c+1-q} \tag{4}$$

Obviously, $P_c(k) > P_1(k)$, i.e. more colluding attackers, better chance to guess k prime numbers. However, the chance is just slightly better as shown in Figure 5(b), because $\frac{P_c(k)}{P_1(k)} = \prod_{q=1}^{k} \frac{n'-q}{n'-c+1-q} \approx 1$ when n' is much greater than c. Hence, collusion can only slightly increase the probability to correctly guess the prime numbers.



(a) Computation of a CRT-based token



(b) Length of a CRT-based token

Figure 6: Computation and communication costs

5 Performance Evaluation

In this section, we analyze the performance of the proposed authentication approach in terms of its storage, computation, and communication overhead in the forwarding procedure.

5.1 Storage

In order to generate a CRT-based authentication token, a source node should record all symmetric keys and all prime numbers assigned to the forwarding nodes. For CRT computation, the source also needs to pre-compute and records coefficients c_j. The storage in a source node is thus $O(n)$, where n is the number of hops in the route. The storage in a forwarding node is $O(1)$, because it only needs to store a symmetric key and a prime number that are only shared with the source.

5.2 Computation

For each data packet, the computation of a token in a source depends on the hop number of a route. Both the random number generation and the CRT computation are $O(n)$, proportional to the hop number n. Upon receiving a data packet, a forwarding node should verify both the token. The verification of key consists of the computation of the corresponding random number and two modulo calculations. In a computer with a Pentium IV 2.4GHz CPU, a hash (MD5) can be computed about every $1.55\mu s$. The computation of a CRT-based token is shown in Figure 6(a) (excluding the computation of random numbers). For a 9-hop route, a token can be computed about every $0.56\mu s$ once after 9 hash functions are computed for the generation of 9 random numbers.

5.3 Communication Overhead

The communication overhead is measured as the length of the authentication token carried in each packet. In the CRT-based authentication approach, the length of a token depends on the number of hops and the length of prime numbers. Assume that a route has n hops, each of which is assigned a prime number m_j. The length of a token is $\sum_{j=1}^{n} l(m_j)$, where $l(m_j)$ is the length of m_j in bits. As previously analyzed, the length of prime number determines the probability to filter a junk packet. Therefore, the communication overhead is determined by the security requirement of the network.

For example, if we want that a junk packet can be filtered with a 99% probability in one hop, then a prime number should have at least 8 bits. There are total 533 prime numbers that have 8 to 12 bits. Accordingly, the length of a token (measure in bytes) is depicted in Figure 6(b). The token length is roughly less than 1.5 byte per hop and proportional to the hop number. Obviously, the token is larger when the route needs higher security and has more hops.

5.4 Comparison

In the following, the CRT-based authentication protocol is compared with two approaches in terms of security and cost. One approach is the basic hop-by-hop source authentication protocol presented in Section 2.2.1. The other approach is based on one-way hash chain (OHC). (An overview of one-way hash chain is given in Appendix B.) We assume that a route has n hops, R_j is an attacker, and the OHC is a chain of m hashes. The comparison is summarized in Table 2. The metrics of comparison include security, computation, storage and communication overhead. The security is measured as whether these approaches prevent injection and slander. The storage, the computation and the communication overhead are measured as in previous performance evaluation.

Table 2: Comparison

Defense approach	Security		Source		Forwarding nodes	Communication overhead
	Against injection	Against slander	Storage	Computation	Computation	
CRT-based	Y	Y	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Basic	Y	N	$O(n)$	$O(n)$	$O(1)$	$O(n)$
OHC-based	Y	N	$O(m)$	$O(1)$	$O(1)$	$O(1)$

As shown in Table 2, the CRT-based authentication approach provides security against both injection and slander. As previously analyzed, the basic authentication protocol allows a forwarding node to filter junk packets, but cannot prevent slander. The OHC-based approach can prevent injection as well, because an attacker cannot forge the hash chain. But when a forwarding node receives a packet, the node cannot verify the packet until a later time. Hence, an injected packet cannot be filtered immediately. Consequently, when an injected packet is detected, the defender cannot identify which node injected the packet.

The CRT-based and the basis hop-by-hop source authentication approaches have the same cost, because a CRT-based authentication token is an integration of the individual tokens that are used in the basic authentication protocol. Their cost is determined by the number of hops in a route. A source node in these two approaches pays its resources packet by packet. On the contrary, the cost of the OHC-based authentication approach is determined by the application, in stead of the route. The OHC-based approach needs a source node to pay all resources at the beginning for its following communication. The source node needs to pre-compute and store an entire hash chain, which needs $O(m)$ storage and computation. When the hash chain is exhausted, the source needs to re-compute a new hash chain. Hence, the source node is required to estimate the length of the chain according to the application. If the source overestimates the length of the OHC, its resource is wasted on some pre-computation for the unused part of the OHC. If the length is underestimated, the OHC may frequently re-compute hash chains. Compared to OHC, the CRT-based approach is more flexible for on demand data forwarding, because it only requires a sender node to generate an authentication token on demand.

6 Related Works

6.1 Source Authentication

To identify forged or spoofed packets, source authentication is generally used because the cryptographic technique ensures only the real source can send out legitimate packets. In the literature, research in the fields of multicast source authentication, broadcast source authentication, and hop-by-hop source authentication provides many candidate solutions to defend against packet injection attacks.

In multicast source authentication, multiple receivers can verify whether the received data originated from the claimed source and was not modified en route. The recent research efforts [16, 25] in multicast source authentication focused on amortizing the cost of a digital signature over multiple packets. The source amortizes signature over the message digests of a block of packets, and the receiver can verify all the packets in a batch. Although this scheme has the lowest amortized packet overhead, it needs an en route node to receive the whole block for verification and does not tolerate any packet losses. Some researches proposed techniques that do tolerate packet loss by using expanded graph [23], authentication chain [6], distillation code [11] or erasure code [17]. 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 [20] based on one-way key chain, which is also used in securing routing protocols [7, 8]. A one-way key chain is a sequence of keys $k_0, k_1, ..., k_n$, where $k_i \leftarrow H(k_{i+1})$ and $H(*)$ is a hash function. When the source sends data packets, it first computes such a chain, and then sends k_i with the i^{th} packet. A receiver can verify k_i by computing whether $k_{i-1} = H(k_i)$, where k_{i-1} is the key in the previous packet. To start the scheme, a sender uses a regular signature scheme to sign the initial key. All subsequent packets are authenticated through the one-way key chain. TESLA is efficient in computation and can tolerate the loss in the following data packets. However, this scheme requires that the source computes the chain in advance and that the nodes buffer the packets in order to verify them later. However, this scheme cannot guarantee to filter a forged packet instantly before other nodes to forward it to the next hop. Therefore, it is not suitable for this study.

Hop-by-hop source authentication approaches [26, 29] have been proposed for filtering injected false data packets in sensor networks. In [26], Ye *et al* propose a statistic filtering scheme that allows enroute nodes to filter out false data packets with some probability. In each packet, the scheme attaches token designated to nodes between the source and the sink, so that a packet will be verified with a high probability before it reaches the sink. In [29], 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. In these two schemes, the sender is actually a cluster of nodes which each contributes to a data packet if they agree on the same event. Thus these schemes cannot be applied directly in our setting. Hop-by-hop source authentication provides two most desired defense features: (1) immediate source authentication and (2) inherent support of on-demand applications in ad hoc networks. However, they still cannot help defenders to identify and remove the attackers. Hence, we use them as the basis to develop our new scheme.

6.2 Intrusion Detection

Intrusion detection is also a candidate solution in defending against packet injection attacks. In [28], Zhang *el al.* proposed a general architecture to have all nodes participate in intrusion detection. Each node takes two roles. First, a node needs to monitor transmission in its neighborhood in order to detect misbehavior in its nearby nodes. Then, each node should cooperate with its neighboring nodes to exchange intrusion detection information in order to detect the malicious node. In [15], Marti *et al.* 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 [9].

All these intrusion detection approaches needs 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. Hence, these intrusion detection approaches need are not suitable to address the attack in this paper.

7 Conclusion and Future Work

In this paper, we presented a CRT-based authentication approach for the defense against packet injection DoS attacks in ad hoc networks. The approach not only preserves the filtering of junk packet as in existing authentication protocols, but also help to isolate the attackers with a high probability. A CRT-based authentication token allows forwarding nodes to immediately filter out injected packets with very high probability. It also ensures that defenders can focus on investigating only two nodes to find out the real attacker once failed verifications are detected. The proposed approach is lightweight with storage and computation costs proportional to the number of hops in a route instead of the number of packets sent in the source node. The approach is resistant to colluding attackers.

One of our future work is to study how to investigate on whether or not a report is a hoax. Because a malicious node may report to deceive the defender, our current strategy is to isolate both the reporter and the reportee. However, this makes one good node unable to forward packets. Therefore, we are interested to study how to put evidence in the report, so that the defender can tell whether the report is a hoax or not. If it is not a hoax, we will release the reporter. Otherwise, if it is a hoax, the reportee will be released.

References

[1] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: a taxonomy and some efficient constructions. In *IEEE Infocom*, volume 2, pages 708–716, 1999.

[2] Guang-huei Chiou and Wen-Tsuen Chen. Secure broadcasting using the secure lock. *IEEE Transactions on Software Engineering*, 15(8):929–934, 1989.

[3] 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.

[4] Laurent Eschenauer and Virgil D. Gligor. A key-management scheme for distributed sensor networks. In *ACM CCS*, pages 41–47, 2002.

[5] Rosario Gennaro and Pankaj Rohatgi. How to sign digital streams. In *Advances in Cryptology, CRYPTO*, pages 180–197, 1997.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

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

[14] Haiyun Luo, Jiejun Kong, Petros Zerfos, Songwu Lu, and Lixia Zhang. Self-securing ad hoc wireless networks. In *IEEE Symposium on Computers and Communications*, Italy, 2002.

[15] 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.

[16] Jung Min Park, Edwin K. P. Chong, and Howard Jay Siegel. Efficient multicast packet authentication using signature amortization. In *IEEE Symposium on Security and Privacy*, page 227, 2002.

[17] 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.

[18] 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.

[19] A. Perrig, R. Canetti, D. Tygar, and D. Song. The tesla broadcast authentication protocol. *Cryptobytes*, 5(2):2–13, 2002.

[20] 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.

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

[22] Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *ACM CCS*, pages 93–100, 1999.

[23] 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.

[24] William Stallings. *Cryptography and network security: principles and practice*. 2002.

[25] Chung Kei Wong and Simon S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Transactions on Networking*, 7(4):502–513, 1999.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

[31] X. Zou, B. Ramamurthy, and S. Magliveras. Chinese remainder theorem based hierarchical access control for secure group communications. In *International Conference on Information and Communication Security*, volume 2229, pages 381–385, 2001.

A Chinese Remainder Theorem

In essence, the CRT states that it is possible to reconstruct integers in a certain range from their residues modulo a set of pairwise relatively prime numbers.

Let $m_1, ..., m_k$ be integers that are pairwise relatively prime for $1 \leq i, j \leq k$, that is the greatest common divisor of m_i and m_j is 1, if $i \neq j$. Let M be the product of all the m_is, i.e. $M = \prod_i^k m_i$.

Given any integer A in the range $[0, M-1]$, A can be uniquely represented by a k-tuple, in which $a_i = A \ mod \ m_i$.

$$A \leftrightarrow (a_1, a_2, ..., a_k)$$

In another word, given a k-tuple $(a_1, a_2, ..., a_k)$ and $m_1, ..., m_k$, we can derive a unique integer A, where $c_i = M_i \times (M_i^{-1} \ mod \ m_i)$ and $M_i = \frac{M}{m_i}$. $(M_i^{-1} \ mod \ m_i)$ is computed based on the extended Euclid's algorithm.

$$A = (\sum_{i=1}^{k} a_i c_i) \ mod \ M$$

For example, let $m_1 = 37$, $m_2 = 49$, and thus $M = m_1 m_2 = 1813$, $M_1 = 49$ and $M_2 = 37$. Using the extended Euclid's algorithm, $M_1^{-1} = 34 \ mod \ m_1$ and $M_2^{-1} = 4 \ mod \ m_2$. Let $A = 973$, and thus $a_1 = 11$ and $a_2 = 42$. Hence, 973 is represented as $(11, 42)$. A can also be calculated as

$$\begin{aligned} A &= (a_1 c_1 + a_2 c_2) \ mod \ M \\ &= (a_1 M_1 M_1^{-1} + a_2 M_2 M_2^{-1}) \ mod \ M \\ &= [(11)(49)(34) + (42)(37)(4)] \ mod \ 1813 \\ &= 973 \end{aligned}$$

B One-way Hash Chain

One-way hash chain (OHC) [12] was initially proposed by Lamport for password authentication. An OHC is a chain of keys generated through repeatedly applying a one-way hash function H on a random number as follows, where k_Q is a random number and k_0 is securely distributed to all receivers at the beginning.

$$k_i = H(k_{i+1}) \ for \ 0 \leq i \leq Q$$

Then, the source node releases the keys in its OHC chain in the reverse order of their generation, and use the keys to authenticate packets. For example, the i-th packet pkt_i is

$$pkt_i = \{P||H(P)\}_{k_i}||k_{i-1}$$

A packet cannot be verified until the following packet is received. For example, when a node receives pkt_i, it uses pkt_i. to verify pkt_{i-1} and then buffers pkt_i. When it receives pkt_{i+1}, it first checks whether k_i in pkt_{i+1} satisfies $k_{i-1} = H(k_i)$. If yes, k_i is a good key. Then, it uses k_i to verify $\{P||H(P)\}_{k_i}$. If yes, pkt_i passes the verification.

Because the key chain is released in the reverse order of the generation, it is computationally infeasible for any node other than the source to derive the keys in the chain that have not been released yet. This property makes attackers unable to forge the OHC. More details on applying OHC in authentication can be found in [20, 21].