Efficient Security Mechanisms for Routing Protocols

Yih-Chun Hu
Carnegie Mellon University
yihchun@cs.cmu.edu

Adrian Perrig
Carnegie Mellon University
perrig@cmu.edu

David B. Johnson
Rice University
dbj@cs.rice.edu

Abstract

As our economy and critical infrastructure increasingly rely on the Internet, securing routing protocols becomes of critical importance. In this paper, we present four new mechanisms as tools for securing distance vector and path vector routing protocols. For securing distance vector protocols, our hash tree chain *mechanism forces a router to increase the distance (metric) when forwarding a routing table entry. To provide authentication of a received routing update in bounded time, we present a new mechanism, similar to hash chains, that we call* tree-authenticated one-way chains*. For cases in which the maximum metric is large, we present* skiplists*, which provides more efficient initial computation cost and more efficient element verification; this mechanism is based on a new cryptographic mechanism, called MW-chains, which we also present. For securing path vector protocols, our* cumulative authentication *mechanism authenticates the list of routers on the path in a routing update, preventing removal or reordering of the router addresses in the list; the mechanism uses only a single authenticator in the routing update rather than one per router address. We also present a simple mechanism to securely switch one-way chains, by authenticating the next one-way chain using the previous one. These mechanisms are all based on efficient* symmetric *cryptographic techniques and can be used as building blocks for securing routing protocols.*

1. Introduction

Routing protocols are difficult to efficiently secure. An attacker may, for example, attempt to inject forged routing messages into the system, or may attempt to modify legitimate routing messages sent by other nodes. An attacker may also attempt to exploit mechanisms in the routing protocol, such as those intended to quickly spread new routing information, to instead consume large amounts of network and router resources. Even the addition of cryptographic mechanisms to a routing protocol may make the protocol vulnerable to such attacks, since traditional security mechanisms are generally expensive in terms of CPU time; an attacker may be able to cripple several routers simply by flooding each router with large numbers of randomly generated, forged routing messages, which then must be authenticated and rejected by the router, leading to a denial of service by consuming all router CPU time.

Current routing protocols in use in the Internet today, such as the Border Gateway Protocol (BGP) [36] or the Routing Information Protocol (RIP) [23], were originally designed to operate in a trusted environment, assuming no malicious nodes. However, with the growing importance and usage of the Internet, an increasing number of corporations and public services are becoming dependent on the correct functioning of the Internet, and routing protocol security has become a significant issue. The command and control of critical infrastructures (such as the control of the power grid) and the emerging use of the Internet to carry voice traffic are two examples of this trend. The importance of securing Internet routing has also been illustrated by recent BGP misconfigurations [22], in which non-malicious BGP speakers have been able to disrupt Internet routing as a result of incorrect configuration.

Several researchers have proposed secure network routing protocols, but most have used standard digital signatures to authenticate routing update messages [18, 19, 20, 32, 38, 39, 40]. Similarly, in the area of secure multihop wireless *ad hoc network* routing, most researchers use standard digital signatures to authenticate routing messages [5, 41, 43]. Unfortunately, generation and verification of digital signatures is relatively inefficient, and it is thus challenging to design a scalable, efficient, and viable secure routing protocol based on such *asymmetric* cryptography.

Symmetric cryptographic primitives are much more efficient than asymmetric primitives, but so far, few security mechanisms based on symmetric cryptography have been designed for the requirements of routing protocols. We now discuss the exceptions of which we are aware.

Three mechanisms based on symmetric cryptography have been proposed to secure link state routing updates. Cheung [2] presents an efficient time-based authentication protocol to authenticate link state routing updates. The

This work was supported in part by NSF under grant CCR-0209204, by NASA under grant NAG3-2534, and by gifts from Schlumberger and Bosch. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, NASA, Schlumberger, Bosch, Rice University, Carnegie Mellon University, or the U.S. Government or any of its agencies.

proposed authentication is optimistic, though, and routers use the routing update before it is authenticated. Hauser, Przygienda, and Tsudik [10] propose to use efficient one-way hash chains to authenticate link state routing updates. Zhang [42] subsequently improves their mechanism and presents a chained Merkle-Winternitz one-time signature [27, 26], similar to our basic MW chains scheme that we present in Section 4.6, although our technique is more space-efficient.

Heffernan [12] assumes that neighboring routers share secret keys, and routers use MD5 to authenticate each other's messages. This approach allows BGP to protect itself against the introduction of spoofed TCP segments into the connection stream (TCP resets are of particular concern).

Basagni et al. [1] present a protocol with a network-wide shared key for use in routing, purely based on symmetric cryptography. However, their approach assumes secure hardware to protect the key.

We have previously developed two efficient secure routing protocols based on symmetric cryptography. Our SEAD protocol [13] introduces a new efficient mechanism, based on one-way hash chains, to secure distance vector routing updates. Our Ariadne routing protocol [14] is a secure on-demand ad hoc network routing protocol using source routing.

In this paper, we present four new security mechanisms based on efficient *symmetric* cryptographic techniques, that can be applied to strengthen current distance vector and path vector routing protocols or can be incorporated into the design of new secure routing protocols. For securing distance vector protocols, our *hash tree chain* mechanism forces a router to increase the distance (metric) when forwarding a routing table entry. To provide authentication of a received routing update in bounded time, we present a new mechanism, similar to hash chains, that we call *tree-authenticated one-way chains*. For cases in which the maximum metric is large, we present *skiplists*, which provides more efficient initial computation cost and more efficient element verification; this mechanism is based on a new cryptographic mechanism, called MW-chains, which we also present. For securing path vector protocols, our *cumulative authentication* mechanism authenticates the list of routers on the path in a routing update, preventing removal or reordering of the router addresses in the list; the mechanism uses only a single authenticator in the routing update rather than one per router address. We also present a simple mechanism to securely switch one-way chains, by authenticating the next one-way chain using the previous one.

In Section 2 of this paper, we discuss the assumptions underlying our secure mechanisms. Section 3 describes the basic cryptographic mechanisms that we use. Section 4 describes our new mechanisms for improving security in distance vector protocols, and Section 5 presents our new mechanisms for building efficient and secure path vector routing protocols. Finally, we present our conclusions in Section 6.

2. Assumptions

In designing our mechanisms to build secure routing protocols, we make the following assumptions on node capability and key setup.

2.1. Node Assumptions

The computational resources of network nodes vary greatly, from a high-end Internet backbone router to a tiny ad hoc network node. To make our results as general as possible, we design our mechanisms to be extremely lightweight and efficient. This allows our mechanisms to be used on resource-constrained ad hoc network nodes and enables large Internet routers to scale to high bandwidth links. In particular, we design our mechanisms from purely *symmetric* cryptographic functions, such as message authentication codes (MACs) or cryptographic hash functions. In contrast, mechanisms based on asymmetric cryptography are often 3 to 4 orders of magnitude slower than hash functions.

Most previous work on secure Internet and ad hoc network routing relies on *asymmetric* cryptography [18, 20, 32, 38, 39, 40]. However, computing such signatures on resource-constrained nodes is expensive, and such high overhead computations may hinder the routing protocol's scalability to large networks.

We do not assume trusted hardware such as tamperproof modules. Secure routing with trusted hardware is much simpler, since node compromise is assumed to be impossible.

2.2. Security Assumptions and Key Setup

We assume a mechanism that enables the secure and authentic distribution of keying material. Most of our mechanisms require the distribution of authentic public values to enable authentication of subsequent values. However, the cumulative authentication mechanism assumes pairwise shared secret keys, or authentic public keys of other nodes if a broadcast authentication system such as TESLA [34, 35] is used.

Digital signatures and a public-key infrastructure may be used to set up the authenticated public values, as well as to establish pairwise shared secret keys if used in conjunction with a key agreement protocol such as Diffie-Hellman [6].

We assume protection against the immediate replay of routing packets. In a wired network or a static wireless network, each router can be configured with a list of possible neighbors; a router that receives an update from a node not on this list can silently discard that update. In mobile

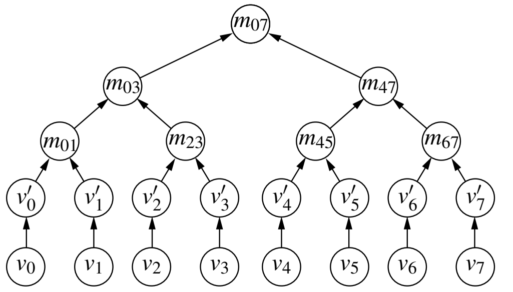

Figure 1: Tree authenticated values

wireless networks, such as ad hoc networks, we have developed *packet leashes* which restrict such immediate replay [15]. In this paper, we assume that one of these mechanisms is used.

3. Cryptographic Mechanisms

In this section, we review the basic cryptographic mechanisms that we use in this work. We first review tree-authenticated values, also known as Merkle hash trees [25]. We also review one-way hash chains, a frequently used cryptographic primitive.

3.1. Tree-Authenticated Values

The mechanism of *tree-authenticated values* is an efficient hash tree authentication mechanism, first presented by Merkle and also known as Merkle hash trees [25]. To authenticate values $v_0, v_1, \ldots, v_{w-1}$, we place these values at the leaf nodes of a binary tree. (For simplicity we assume a balanced binary tree, so w is a power of two.) We first blind all the v_i values with a one-way hash function H to prevent disclosing neighboring values in the authentication information (as we describe below), so $v'_i = H[v_i]$. We then use the Merkle hash tree construction to commit to the values $v'_0, \ldots, v'_{w-1}$. Each internal node of the binary tree is derived from its two child nodes. Consider the derivation of some parent node m_p from its left and right child nodes m_l and m_r: $m_p = H[m_l \mid\mid m_r]$, where $\mid\mid$ denotes concatenation. We compute the levels of the tree recursively from the leaf nodes to the root node. Figure 1 shows this construction over the eight values $v_0, v_1, \ldots, v_7$, e.g., $m_{01} = H(v'_0 \mid\mid v'_1)$, $m_{03} = H[m_{01} \mid\mid m_{23}]$.

The root value of the tree is used to commit to the entire tree, and in conjunction with additional information, it can be used to authenticate any leaf value. To authenticate a value v_i the sender discloses i, v_i, and all the sibling nodes of the nodes on the path from v_i to the root node. The receiver can then use these nodes to verify the path up to the root, which authenticates the value v_i. For example, if a sender wants to authenticate key v_2 in Figure 1, it includes the values v'_3, m_{01}, m_{47} in the packet. A receiver with an authentic root value m_{07} can then verify that

$$H\left[H\left[\ m_{01} \mid\mid H\left[\ H[\ v_2\ ] \mid\mid v'_3\ \right]\right] \ \mid\mid m_{47} \right]$$

equals the stored root value m_{07}. If the verification is successful, the receiver knows that v_2 is authentic.

The extra $v'_0, v'_1, \ldots, v'_7$ in Figure 1 are added to the tree to avoid disclosing (in this example) the value v_3 for the authentication of v_2.

3.2. One-Way Hash Chains

One-way hash chains, or simply *one-way chains*, are a frequently used cryptographic primitive in the design of secure protocols. We create a one-way chain by selecting the final value at random, and repeatedly apply a one-way hash function H. (In our description, we discuss the one-way chain from the viewpoint of usage, so the first value of the chain is the last value generated, and the initially randomly chosen value is the last value of the chain used.) One-way chains have two main properties (assuming H is a cryptographically secure one-way hash function):

- Anybody can authenticate that a value v_j really belongs to the one-way chain, by using an earlier value v_i of the chain and checking that $H^{j-i}(v_j)$ equals v_i.
- Given the latest released value v_i of a one-way chain, an adversary cannot find a later value v_j such that $H^{j-i}(v_j)$ equals v_i. Even when value v_{i+1} is released, a *second pre-image collision resistant hash function* prevents an adversary from finding v'_{i+1} different from v_{i+1} such that $H[v_{i+1}]$ equals v_i.

These two properties result in authentication of one-way chain values: if the current value v_i belongs to the one-way chain, and we see another value v_j with the property that $H^{j-i}(v_j)$ equals v_i, then v_j also originates from the same chain and was released by the creator of the chain.

Jakobsson [16] and Coppersmith and Jakobsson [3] propose a storage-efficient mechanism for one-way chains, such that a one-way chain with N elements requires only $O(\log(N))$ storage and $O(\log(N))$ computation to access an element.

4. Mechanisms for Securing Distance Vector Protocols

In this section, we first review distance vector routing protocols, and then discuss attacks, previous work on securing distance vector routing, and the remaining research challenges. We then present new mechanisms to address these challenges.

The utility of the mechanisms we present is not limited to routing protocols. In particular, the *skiplists* mechanism we present in Section 4.7 allows highly efficient generation and verification of elements in long hash chains, giving a constant factor speedup in both generation and verification. Skiplists are thus particularly useful for protocols that use long one-way hash chains, such as TESLA [34, 35] or BiBa [33].

4.1. Overview of Distance Vector Routing

A *distance vector* routing protocol finds shortest paths between nodes in the network through a distributed implementation of the classical Bellman-Ford algorithm. Distance vector protocols are easy to implement and are efficient in terms of memory and CPU processing capacity required at each node. A popular example of a distance vector routing protocol is RIP [11, 23], which is widely used in IP networks of moderate size. Distance vector routing can also be used for routing within a multihop wireless ad hoc network by having each node in the network act as a router and participate in the routing protocol [17, 30, 31].

In distance vector routing, each router maintains a routing table listing all possible destinations within the network. Each entry in a node's routing table contains the address (identity) of some destination, this node's shortest known distance (usually in number of hops) to that destination, and the address of this node's neighbor router that is the first hop on this shortest route to that destination; the distance to the destination is known as the *metric* in that table entry. When routing a packet to some destination, the node transmits the packet to the indicated neighbor router for that destination, and each router in turn uses its own routing table to forward the packet along its next hop toward the destination.

To maintain the routing tables, each node periodically transmits a routing update to to each of its neighbor routers, containing the information from its own routing table. Each node uses this information advertised by its neighbors to update its own table, so that its route for each destination uses as a next hop the neighbor that claimed the shortest distance to that destination; the node sets the metric in its table entry for that destination to 1 (hop) more than the metric in that neighbor's update.

4.2. Attacks to Distance Vector Routing

Without security mechanisms, a distance vector routing protocol may be vulnerable to a number of attacks from malicious routers. For example, a malicious router could perform the following types of attacks [8, 14, 18, 20, 21, 28, 38, 40]:

- Try to attract as much traffic as possible by advertising short distances to all destinations. This attack is sometimes referred to as a *blackhole attack* [14]. The blackhole attack is quite powerful, because the malicious router can control any communication that passes through it. The router can eavesdrop on data, selectively drop or alter packets, or inject packets. Otherwise eavesdropping on a specific target may be challenging in in today's Internet, as one would need to access the network link to the target, but by claiming a short distance to the target, a malicious router can attract the target's traffic.
- Try to claim longer distances. This attack results in less traffic flowing across the attacking router, potentially allowing the attacker to obtain network services without (for example, in the case of a multihop wireless ad hoc network) using its own network bandwidth and battery power to forward packets from other nodes. We do not attempt to prevent this attack, since it is weaker than a malicious router arbitrarily discarding, delaying, or reordering packets sent to it for forwarding.
- Inject routing loops. This is a powerful attack if the resulting routing loop does not go through the malicious router, since a single packet sent along that loop can cause the packet to be forwarded a large number of times. However, if the routing loop goes through a malicious router, this attack is equivalent to a data flooding attack: for example, if an attacker forms a loop from itself through routers $n_1, n_2, \dots, n_x$ and back to itself, then each time it forwards the packet around the loop, it is equivalent to sending a packet traversing x routers.
- Inject a large number of route updates, to consume network bandwidth and router processing time.

Most of these attacks are forms of denial-of-service attacks, either consuming network resources or preventing packet delivery. Of these attacks, the most powerful attacks are the blackhole attack and the routing loop attack (when the attacker is not in the loop). As a result, we aim to prevent a malicious router from claiming a shorter distance to a target than it actually has, and to prevent a loop from forming such that an attacker is not in the loop.

4.3. Overview of Previous Work on Securing Distance Vector Routing and the Remaining Challenges

We briefly reviewed previous work on secure routing in Section 1. SEAD is a recent secure distance vector routing protocol we designed, that is particularly efficient because it uses one-way hash chains and no asymmetric cryptography. We briefly overview SEAD and discuss the remaining research challenges.

SEAD, the Secure Efficient Ad hoc Distance vector routing protocol [13], is based on based on DSDV [30], and we secure SEAD using hash chains. As in other distance vector protocols such as RIP [11, 23], SEAD requires some limit on hop count (metric), denoted as $k-1$. A distance vector update originated from a node in SEAD contains a *sequence number* and a *metric* for each destina-

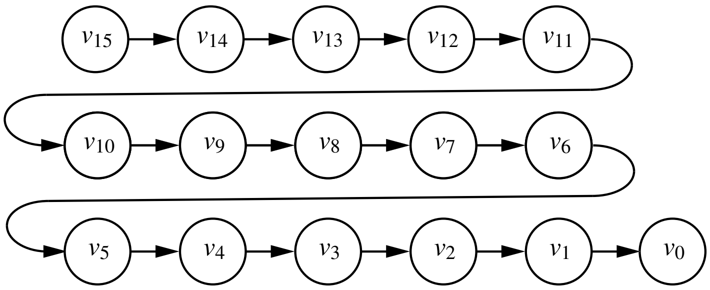

Figure 2: An example of a SEAD hash chain, with $k = 5$ and $n = 3$. Arrows show the direction of hash chain generation; hash chain usage proceeds in the opposite direction.

tion. The sequence number is used to indicate the freshness of each route update. The metric is the distance, measured in number of hops, from the originating node to the destination. When a node receives a route update, for each entry in the route update, it accepts the entry if the entry has a higher sequence number, or if the entry has equal sequence number and a lower metric than the route entry currently in the node's route table for that destination. In order to prevent attacks on the route updates, the sequence number and metric must both be authenticated.

SEAD authenticates the sequence numbers and metrics in route updates using one-way hash chains. As outlined in Section 4.2, we are mainly concerned about the authenticity of a route update, and that a node cannot make a route shorter (in order to prevent the blackhole attack). To initialize, each node N forms a one-way hash chain $v_{kn}, v_{kn-1}, \ldots, v_0$, with $v_{i-1} = H[v_i]$, as we describe in Section 3.2, where $k-1$ is the maximum hop count, and n is the maximum sequence number this hash chain allows. These values are used to authenticate routing update entries that specify this node N as a destination. To allow values $v_{kn}, v_{kn-1}, \ldots, v_1$ to be authenticated, an authentic v_0 is published as an authenticated seed value for the node N, for example by a Certificate Authority. The value v_{ki+j} will be used to authenticate a route update entry with sequence number i and metric $k-j$ for the node N as a destination when $1 \leq j \leq k$.

For example, a hash chain for a node N is depicted in Figure 2, where $k = 5$ and $n = 3$. To initialize, v_0 is published as an authenticated seed value for node N. To start the first route update for entries with N as the destination, the node N first sends v_5 as an authenticator for sequence number 0 and metric 0. A recipient would first authenticate v_5 using the public authenticated value v_0, and then compute v_4 from v_5; the node would then advertise sequence number 0 and metric 1 using authenticator v_4. Similarly, recipients of that update would advertise sequence number 0 metric 2 using authenticator v_3, and so forth. The next time the node N starts route updates for entries with N as the destination, it would disclose v_{10} to authenticate sequence number 1 and metric 0.

Because of the properties of a one-way hash chain, a node cannot forge a routing update with higher sequence number, nor can it forge a routing update with an equal sequence number and lower metric. Larger sequence numbers take precedence over smaller ones, so nodes would simply drop updates with smaller sequence numbers, even if the metric is lower.

Since a node selects the next-hop router towards a destination to be the source address of the routing update with the shortest distance it receives, the source address of each routing update must be authenticated. This authentication can be achieved with a signature, broadcast authentication, or pairwise shared keys. SEAD specifies the use of pairwise shared keys, which exploits the periodic nature of the protocol. When two nodes A and B move within range, one of the two nodes (for example, A) will hear an update sent by the other (B). That node A can begin including in each of its updates symmetric authentication to the new neighbor B. Conversely, when B hears one of A's updates, it will respond by including in B's updates symmetric authentication to its new neighbor A. When the two nodes move away from each other such that they are no longer neighbors, their routing tables will reflect that. For example, when A and B move apart, A's routing table will show that B is no longer a neighbor, as an update with a fresh sequence number will override an older update received directly from B. A can then stop including in its routing updates symmetric authentication to node B.

Although SEAD does prevent a number of attacks, some attacks and shortcomings remain:

- SEAD does not prevent *same-distance fraud*: that is, a node receiving an advertisement for sequence number s and distance (metric) d can re-advertise the same sequence number s and distance d. Section 4.4 presents an approach that prevents this same-distance fraud.
- Another drawback of SEAD is that an attacker can force a victim node to verify a hash chain as long as

$O(ks)$, where k is the maximum number of hops and s is the maximum number of sequence numbers represented by a hash chain. Section 4.5 describes the tree-authenticated one-way chains mechanism, which bounds this effort by $O(k + \lg s)$. The same scheme prevents the sequence number rushing attack, which we present in Section 4.5.

- The overhead to verify authentication values can be large if a node has missed several routing updates. An attacker can exploit this overhead to perform a denial-of-service attack by sending bogus routing updates, forcing a node to spend considerable effort verifying the authenticity. In Section 4.6, we introduce a novel authentication scheme that is a hybrid between a one-way chain and a one-time signature which we call an MW-chain. Based on the MW-chain, we introduce in Section 4.7 a one-way chain that is very efficient to verify in case of missed routing updates. In a network with maximum diameter k, this approach reduces the verification overhead to $O(c\sqrt[c]{k} + \lg s)$ for arbitrary positive integers c. Finally, we reduce the overhead of setting up a single hash chain from $O(ks)$ to $O(s)$.

We now discuss mechanisms that can solve these remaining challenges. Our mechanisms can be generalized to secure many other distance vector protocols.

4.4. Hash Tree Chains for Preventing Same-Distance Fraud

We present an alternative called *hash tree chains* to one-way hash chains for authenticating the distance metric in distance vector protocols, to prevent the same-distance fraud attack introduced above. Our new mechanism forces a node to increase the distance to the destination when sending a routing update. As noted in Section 2, we use packet leashes [15] to prevent an adversary from replaying a routing update in wireless networks, so that the adversary would be a "stealth node" on that route. The packet leash also provides hop-by-hop authentication, preventing an adversary from impersonating another node.

To prevent same-distance fraud, we need to prevent an attacker from replaying the same hash value (thus without increasing the metric) but replacing the node id with the attacker's node id. We construct a special one-way chain, which we call a hash tree chain, where each element of the chain encodes the node id, thus forcing a node to increase the distance metric if it wants to encode its own id. Each step in this one-way chain contains a collection of values, one or more of which are used to authenticate any particular node. These values are authenticated using a Merkle tree, and the root of that Merkle tree is used to generate the collection of values in the next step.

A hash tree chain is a hybrid between a hash tree and a one-way chain. The one-way chain property is used in the same way as in SEAD (to enforce that nodes cannot decrease the distance metric), and the hash tree property is used to authenticate the node id. We construct the hash tree between each pair v_{i-1}, v_i of one-way chain values as follows. From the value v_i, we derive a set of values $b_0, \ldots, b_n$, using a one-way hash function H as $b_j = H[v_i \mid\mid j]$, for each j. We then build a hash tree above those values as described in Section 3.1. The root of the tree becomes the previous value of the one-way chain $v_{i-1} = b_{0n}$. Figure 3 shows an example. The node with the id 1 forwards the shaded values b'_0, b_1, and b_{23} to the neighboring nodes, which can compute the one-way hash tree chain forward to verify the authenticity of values b'_0, b_1, and b_{23}, and use the value b_{03} to sign their own id when forwarding the route update, thus automatically increasing the distance metric.

We now present two examples of how the hash tree chain can be used: when a single value corresponds to a node, and when a γ-tuple of values corresponds to a node. For notational and analytic convenience, we describe hash tree chains for which the number of values between each hash chain value is a power of two.

In a small network, each value b_j can correspond to a single node; since no two nodes share a single value, an attacker has no way to derive its value from the advertisements of neighboring nodes, and hence it must follow the hash tree chain to the next step in order to provide a valid authenticator.

In larger networks, with n nodes, the $O(n)$ overhead of generating each step of the chain may be too great; as a result, we authenticate each node with a γ-tuple of values. Although two nodes share the same γ-tuple of values, an attacker could learn each of its γ values from different neighbors that advertise the same metric, and could then forge an advertisement without increasing the metric. We show that an attacker's probability of success may be sufficiently small. We also change the encoding of a node id for each update, so that an attacker in a static network cannot continue to forge updates once it finds an appropriate set of values from its neighbors. Consider a hash tree chain with 2^m values in each step (and thus a hash tree of height $m+1$). For example, if each node has a unique node id between 0 and $\binom{2^m}{\gamma} - 1$, then the γ-tuple encodes $x = (\text{node id} + H[\text{sequence number}]) \mod \binom{2^m}{\gamma}$, such that the γ-tuple changes for each sequence number.

We now analyze the security of hash tree chains as the probability that a malicious node can forge an advertisement based on the advertisements from its neighbors. Clearly, if each value corresponds to a single node id, no forgery is possible. We now consider the case in which a pair of values (i.e., $\gamma = 2$) represents each node. For our analysis, we consider a hash tree chain with 2^m values at each

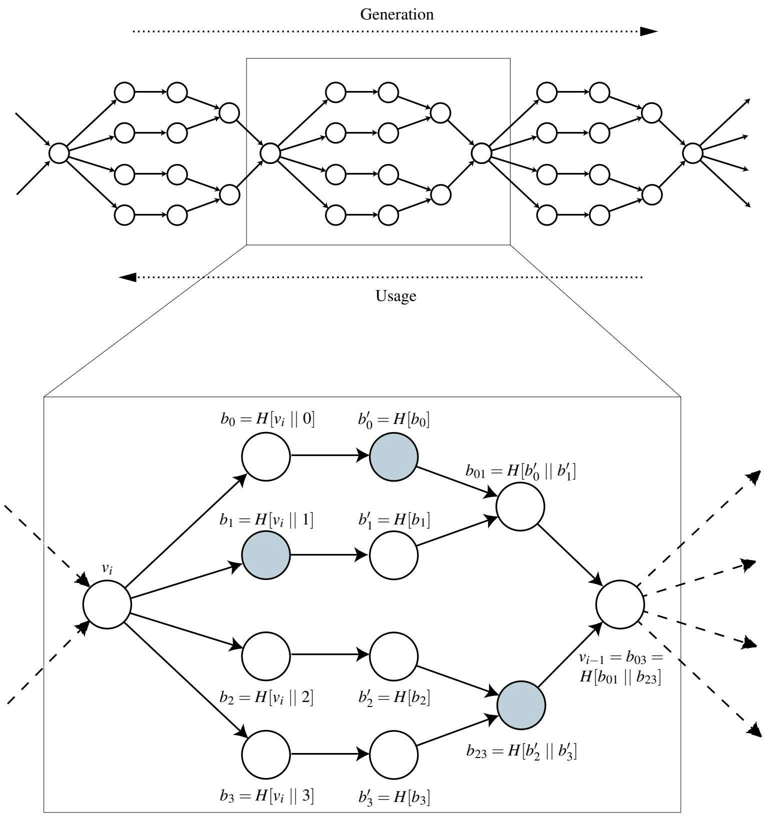

Figure 3: Authenticating one distance metric within a sequence of a hash tree chain. In this example, each element b_i stands for one router, so this hash tree chain supports 4 routers.

step, used in a network with $n = \binom{2^m}{2}$ nodes. We compute the probability that an attacker can claim the same metric after it has heard the same metric advertised from q other nodes. For each of the two values the attacker must produce, there are $2^m - 2$ other nodes that have that particular value. It follows that the attacker has $\binom{(n-1)-(2^m-2)}{q} = \binom{n-2^m+1}{q}$ ways of failing to get a predetermined one of the two values. We now compute the probability that the attacker is unable to obtain either value. Since the set of nodes from which an attacker can receive either value are disjoint, there are $2(2^m - 2)$ nodes that have one of those two values. As a result, the attacker has $\binom{(n-1)-2(2^m-2)}{q} = \binom{n-2^{m+1}+3}{q}$ ways of failing to get either of the two values. Applying the inclusion-exclusion principle, we now compute the number of ways the attacker can fail to obtain both values it needs: $2\binom{n-2^m+1}{q} - \binom{n-2^{m+1}+3}{q}$, of $\binom{n-1}{q}$ possible distributions. The probability of successful defense, then, is

$$\frac{2\binom{n-2^m+1}{q} - \binom{n-2^{m+1}+3}{q}}{\binom{n-1}{q}} \quad .$$

For example, when $m = 8$, then $n = 32640$, and an attacker that hears $q = 3$ advertisements has a 0.000361 (3.61×10^{-4}, or 1.49×2^{-12}) probability of forging a valid authenticator from the three advertisements, without increasing the distance to the destination. In other words, an attacker can decrease its advertised metric by 0.00036 in expectation, or on average once every 2752 rounds. To improve security and reduce route oscillations, we can also require that a node advertise a particular metric for several consecutive sequence numbers before we choose that route. For example, if we require a route to be advertised three consecutive times before we accept it, and if routing updates are sent (and accepted if valid) once per second, then the attacker can successfully send a forged routing update on average only once in over 660 years, given the parameters of n, m, and q above.

To generalize our analysis, we consider the security of the hash tree chain scheme, where a node corresponds to a set of β values. First, we consider the number of ways that an attacker can fail to obtain a specific set of γ different values. There are $\binom{2^m-\gamma}{\beta}$ nodes that do not help the attacker, so there are a total of $\binom{\binom{2^m-\gamma}{\beta}}{q}$ ways to pick q nodes that do not help the attacker.

Let A_i be the set of combinations of nodes that do not include value b_i needed by the attacker. The attacker, then,

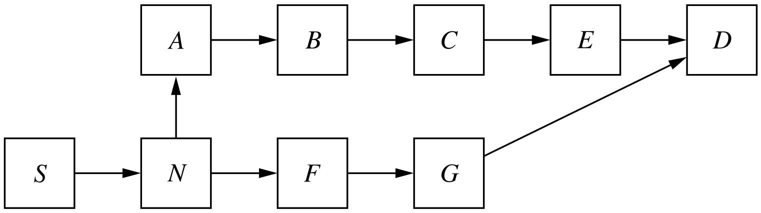

Figure 4: Sample network to demonstrate the sequence number rushing attack.

has $|\cup_{i=1}^{\gamma} A_i|$ ways to fail. We now apply the inclusion-exclusion principle:

$$\begin{aligned}\left|\bigcup_{i=1}^{\gamma} A_i\right| &= \sum_i |A_i| - \sum_{i_1,i_2} |A_{i_1} \cap A_{i_2}| + \cdots + (-1)^{\gamma+1} \left|\bigcap_{i=1}^{\gamma} A_i\right| \\ &= \sum_{i=1}^{\gamma} (-1)^{i+1} \binom{\gamma}{i} \binom{\binom{2^m-i}{\gamma}}{q}\end{aligned}$$

Then the probability of a successful defense is

$$\frac{\sum_{i=1}^{\gamma} (-1)^{i+1} \binom{\gamma}{i} \binom{\binom{2^m-i}{\gamma}}{q}}{\binom{\binom{2^m}{\gamma}-1}{q}}$$

We can now use this to analyze variants of the scheme described earlier. In particular, we look for $n > 32000$.

When $m = 6$, $\gamma = 3$. (This represents a four-fold reduction in computation, in exchange for a 17% increase in overhead). Using $q = 3$ as before, an attacker has a 1.675×10^{-3} probability of success; when three consecutive advertisements are required for the same metric before a routing change is made, the attacker succeeds once every 6.74 years.

When $m = 5$, $\gamma = 4$. (This represents a eight-fold reduction in computation, in exchange for a 33% increase in overhead). Using $q = 3$ as before, an attacker has a 8.023×10^{-3} probability of success; when four consecutive advertisements are required for the same metric before a routing change is made, the attacker succeeds once every 7.65 years.

4.5. Tree-Authenticated One-Way Chains for Preventing the Sequence Number Rushing Attack

In protocols such as SEAD, a node that has missed a number of sequence numbers may need to perform a large number of hash operations to bring its chain up-to-date. This creates a potential denial-of-service vulnerability: if an attacker knows that a victim missed several recent updates for a destination, the attacker can flood the victim with updates containing recent sequence numbers but bogus authenticators; the victim must then perform many hash operations for each update received in an attempt to verify each update. Alternatively, the attacker can fabricate an update with sequence numbers far in the future, thus requiring each node receiving such an update to perform a large number of hash operations to determine that the update is bogus, although this attack can be somewhat mitigated using loose time synchronization, and rate limiting the use of new sequence numbers.

Another attack is the *sequence number rushing attack*. We explain this attack with an example. Consider the case in which a malicious node A tries to attract traffic flowing from a source S to a destination D through S's neighbor N. Figure 4 shows the network setup. Let the attacker A be 4 hops from D, and N be 3 hops from D. If A hears new routing updates from D before they reach N, A can rush the routing update to N. If we use the policy that a node always uses the routing update with the most recent sequence number, N will forward traffic from S to D through A until it hears the routing update with the new sequence number from F which contains a shorter route. To remedy this rushing attack, we adapt a *delayed route update adoption policy*: always use the shortest route from the previous sequence number. For example, when node N hears the first routing update with sequence number i for destination D, it will use the shortest update of sequence number $i-1$. Unfortunately, this approach is still vulnerable to an attack in which A sends two routing updates to N after it hears the update for sequence i: it forges an update with distance 0 of sequence number $i-1$, followed by an update of distance 3 for sequence number i. The tree-authenticated one-way chain mechanism we present in this section prevents A from forging low distance metrics for previous route updates. Together with the delayed route update adoption policy, we can prevent the sequence number rushing attack.

We describe here our efficient tree-authenticated one-way chain mechanism, which has two properties in addition to those of the hash chain in SEAD: first, it bounds the effort to verify an update; and second, it prevents a node with fresh sequence number information from fabricating lower metric authenticators for old sequence numbers.

In our new scheme, we use a new hash chain for each sequence number. A node using this scheme generates a random hash chain root $h_{0,s}$ for each sequence number s,

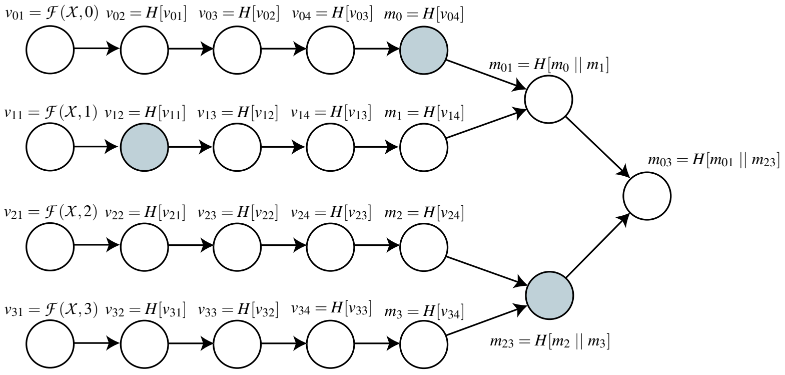

Figure 5: Example tree-authenticated one-way chain construction for authenticating a sequence of one-way chains. The instance in this figure allows 4 sequence numbers to be authenticated, and metrics up to 3. The shaded values represent sequence number 1 metric 1.

for example by using a PRF $\mathcal{F}$ and a secret master key $\mathcal{X}$ to derive $h_{0,s} = \mathcal{F}(\mathcal{X}, s)$. Given the authentic anchor of this hash chain $h_{k,s} = H^k[h_{0,s}]$ (where k is the maximum metric), any node can authenticate $h_{m,s}$, which is the authenticator for sequence number s and metric m.

To allow nodes to authenticate these anchors $h_{k,s}$, each node builds a hash tree, using the hash chain anchors as leaves (Section 3.1). When a node sends an update with a new sequence number s, it includes the root of the hash chain $h_{0,s}$, the anchor of the hash chain $h_{k,s}$, and the path to the root of the hash tree. To authenticate any update, the node verifies the anchor by following the path to the root of the hash tree. It then verifies the hash value $h_{m,s}$ by verifying that $h_{k,s} = H^{k-m}[h_{m,s}]$. Since the maximum hash chain length is k and the anchor verification requires $O(\log(s))$ effort, where s is the number of sequence numbers represented by any root, the computation required to verify any update is bounded by $k + \log(s)$.

4.6. The MW-Chains Mechanism

In this section, we present a new cryptographic mechanism, which we use in the next section to improve the efficiency of secure network routing and to prevent a class of denial-of-service attacks. This mechanism is an extension to the Merkle-Winternitz one-time signature construction [26]. That construction was subsequently used and extended by Even, Goldreich, and Micali [7], and by Rohatgi [37]. Our extension to this signature construction, which we call a one-way Merkle-Winternitz chain, or MW-chain, provides instant authentication and low storage overhead. This one-way chain contains a list of values, called *heads*, and between any two heads are a set of *signature branches* and a set of *checksum branches*. To achieve low storage overhead, we derive these branches from a single head using a one-way hash function H.

The most basic way to construct an MW-chain is with one signature branch and one checksum branch between each head. Assuming we want one set of branches to sign up to N values, we choose the length of the signature branch and the checksum branch to be N; that is, we choose $\ell_1 = \ell'_1 = N$. A random value that is ρ bits long is chosen as the first head value v_n. Next, the signature and checksum branches are computed using

$$\begin{aligned} s_{1,\ell_1} &= H[v_i \mid\mid \text{“}s\text{”} \mid\mid 1] \\ s_{1,x-1} &= H[s_{1,x}] \\ c_{1,\ell'_1} &= H[v_i \mid\mid \text{“}c\text{”} \mid\mid 1] \\ c_{1,x-1} &= H[c_{1,x}] \end{aligned}$$

for $2 \leq x \leq \ell_1$. Finally, the next head value is $v_{n-1} = H[s_{1,1} \mid\mid c_{1,1}]$. The signature of a value n using this MW-chain is the ordered set $\{s_{1,n}, c_{1,N-n}\}$. An attacker can produce $s_{1,j}$, for $j < n$, but then cannot produce $c_{1,N-j}$. Similarly, an attacker can produce $c_{1,N-j}$ for $j > n$, but then cannot produce $s_{1,j}$.

More generally, an MW-chain can have m signature branches and m' checksum branches. We call the lengths of the signature branches $\ell_1, \ell_2, \ldots, \ell_m$ and the lengths of the checksum branches $\ell'_1, \ell'_2, \ldots, \ell'_{m'}$. The signature for some value n is the ordered set

$$\{s_{1,n_1}, s_{2,n_2}, \ldots, s_{m,n_m}, c_{1,n'_1}, c_{2,n'_2}, \ldots, c_{m',n'_{m'}}\}$$

where $n_i = \left(\left\lfloor \frac{n}{\prod_{j=1}^{i} \ell_j} \right\rfloor \bmod \ell_i \right) + 1$, and $n'_i = \left(\left\lfloor \frac{\sum_{j=1}^{m} \ell_j - n_j - 1}{\prod_{j=1}^{i} \ell'_j} \right\rfloor \bmod \ell'_i \right) + 1$.

For example, Figure 6 shows an example MW-chain being used to sign the value 58. In this example, there are 3 signature chains, each of length 4, and 2 checksum chains, also each of length 4. To sign the value 58 in this case,

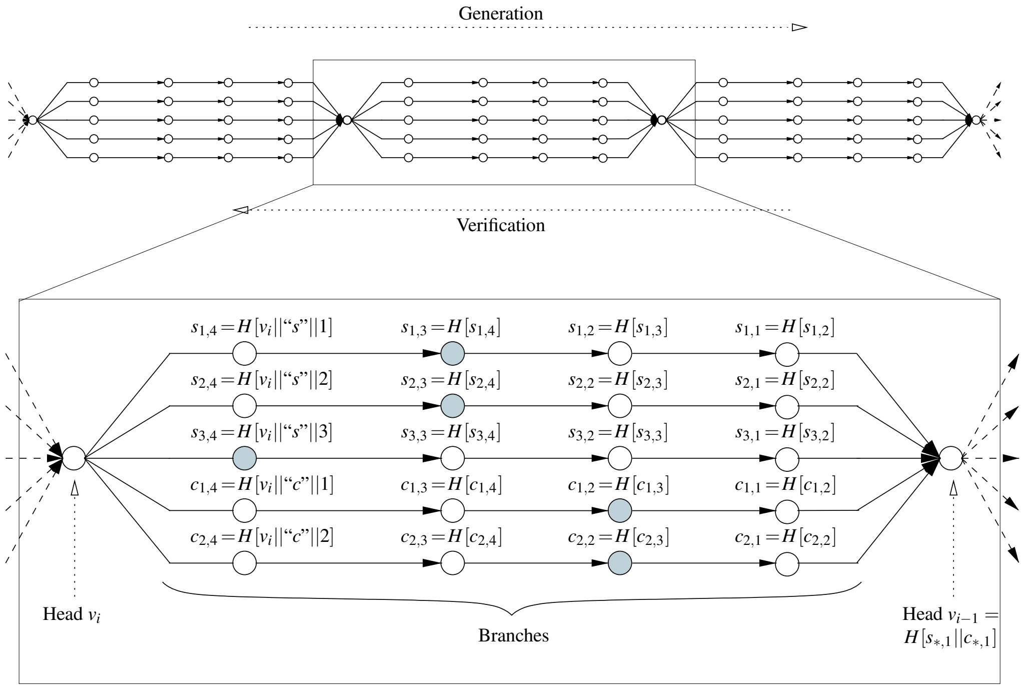

Figure 6: An example MW-chain being used to sign the value 58

$n_1 = (58 \mod 4) + 1 = 3$, $n_2 = \left(\left\lfloor \frac{58}{4} \right\rfloor \mod 4\right) + 1 = 3$, and $n_3 = \left(\left\lfloor \frac{58}{16} \right\rfloor \mod 4\right) + 1 = 4$, so $n'_1 = ((12 - (2+2+3)) \mod 4) + 1 = 2$ and $n'_2 = \left(\left\lfloor \frac{12-(2+2+3)}{4} \right\rfloor \mod 4\right) + 1 = 2$. The signature is thus the ordered set $\{s_{1,3}, s_{2,3}, s_{3,4}, c_{1,2}, c_{2,2}\}$.

Every signature chain i can sign $\log_2(\ell_i)$ bits. All signature chains together can sign $b = \sum_{i=1}^{m} \log_2(\ell_i)$ bits. To sign a message M of b bits, the signer splits the message into m chunks $M_1, \ldots, M_m$, each of size $\log_2(\ell)$ bits. The signer adds the values s_{i,M_i+1} to the signature. Note that the first value of a signature chain signs the number 0, the second value a 1, and so on.

To prevent an attacker from forging a message M', where $M_i \geq M'_i, 1 \leq i \leq m$ (because anybody can compute the one-way chain into that direction to know the previous values) the sender uses a checksum chain that moves in the opposite direction of the signature chains. Consequently, an attacker that tries to sign M' as described above would need to invert the checksum chain, which is computationally infeasible. The checksum chains need to be long enough to sign the maximum sum that might occur in the signature chains: $\prod_{i=1}^{m'} \ell'_i \geq 1 + \sum_{i=1}^{m} (\ell_i - 1)$.

The signer computes the checksum of the signature chains by summing all the values that it signed with the signature chains: $s = \sum_{i=1}^{m} M_i$. The signer splits the checksum into m' checksum chunks. The checksum chunks are encoded in reverse in the checksum chains, compared to how the message chunks are encoded n the signature chains. For checksum chunk s_i, the signer adds the value $c_{i,\ell'-s_i}$ to the signature.

Rohatgi [37] proposes a concrete instantiation to sign an 80-bit message: 20 signature chains of length 16, and 3 checksum chains of length 16. Zhang [42] presents a similar mechanism, except that he does not bring the multiple hash chains together into heads. As a result, MW-chains have an advantage in reduced storage overhead.

In retrospect, it may seem that the development of hash tree chains was unnecessary: a node could use an MW-chain to sign its node identifier, thus preventing a node from directly replaying its authenticator. Unfortunately, using MW-chains in this context is not secure, since an attacker receiving several advertisements of equal metric can recover many values of the signature and checksum chains. For example, we performed a Monte Carlo simulation for a scenario in which $n = 32640$ nodes are represented using 5 signature chains of length 8, and 2 checksum chains of length 4, and each attacker hears 3 advertisements. In this case, an attacker was able to forge a valid signature with a probability of 0.196, in contrast to the hash tree chain, where the probability of successful forgery was 3.6×10^{-4}.

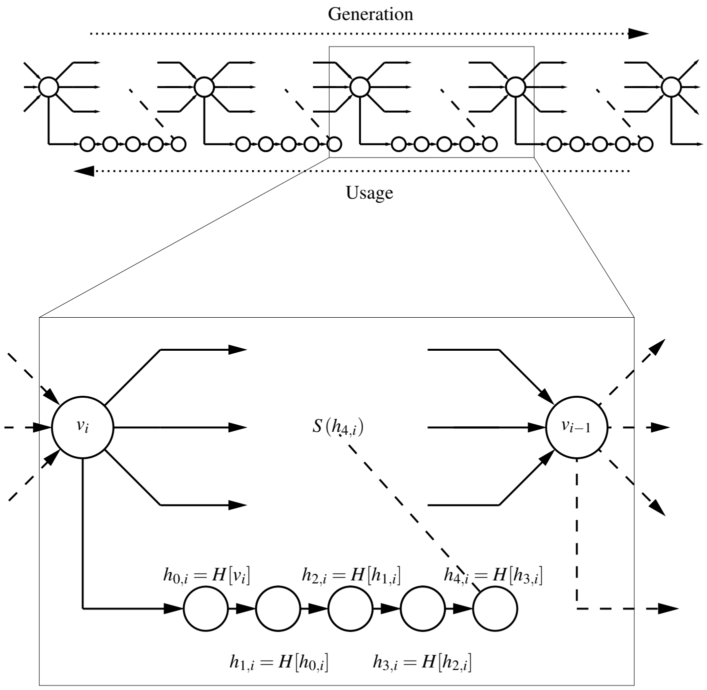

Figure 7: One step in a skiplist with $k = 4$

4.7. Skiplists for Preventing Denial-of-Service Attacks and for Faster Hash Chain Authentication

In Section 4.5, we described a mechanism that allows each node to verify a hash chain without needing to perform a large number of hash functions. However, the amount of effort required to verify an element is $O(k + \lg s)$, where k is the length of the hash chain and s is the number of hash chains. The network overhead is $O(\lg s)$, and initial computation cost is $O(ks)$. If the maximum metric is large, this approach may be prohibitively expensive, either in terms of initial computation cost or for element verification.

In this section, we describe an approach which, when combined with the Merkle tree authentication described in Section 4.5, has $O(c\sqrt[c]{k} + \lg s)$ verification cost and $O(s\sqrt[c]{k})$ generation cost, at the cost of $O(c + \lg s)$ overhead, where c is any positive integer. We achieve this by creating a *skiplist*, which is a chain that, when followed for one step, skips over many steps in a virtual hash chain. In the most basic version, a skiplist is $\sqrt{k}$ long, and each step in a skiplist represents $\sqrt{k}$ steps in a hash chain, which represents $c = 2$. In general, skiplists can be embedded inside skiplists, allowing values of $c > 2$. Skiplists can also be used in protocols such as TESLA [34, 35] and BiBa [33], to improve the efficiency of following long hash chains.

Each skiplist is represented by an MW-chain capable of signing enough bits to ensure security (for example, 80 bits). Each step in this MW-chain represents m steps in a virtual hash chain. To generate the hash chain (or skiplist) associated with this step, a new head is chosen by hashing the head of this step. The anchor of this hash chain (or skiplist) is computed, and that step in the MW-chain is used to sign this new anchor. For example, if the head of one step in a skiplist is v_i, a node forms $h_{0,i} = H[v_i]$, computes the corresponding anchor (for example $h_{m,i} = H^m(h_{0,i})$, if this is the last level of skiplists). It signs this anchor using v_i, as described in Section 4.6.

More concretely, we consider the case in which there is one level of skiplist, and each step in the skiplist corresponds to m steps in the virtual hash chain. If the MW-chain is n steps long, then the virtual hash chain is mn steps long. The leftmost element in this virtual hash chain is v_n, from which all chain elements can be derived. An alternative representation is the pair $(h_{0,n}, S_{v_n}(h_{m,n}))$, where $S_{v_n}(h_{m,n})$ represents $h_{m,n}$ signed using v_n. The next element is the pair $(h_{1,n}, S_{v_n}(h_{m,n}))$. The element at position $(m+1)$ from the left is $(h_{1,n-1}, S_{v_{n-1}}(h_{m,n-1}))$. In general, the xth element from the left is represented by the pair $(h_{x \bmod n, y}, S_{v_y}(h_{m,y}))$, where $y = n - \lfloor \frac{x}{m} \rfloor$.

To verify a hash element, a node follows the hash chain to the anchor, and verifies the signature of the anchor. If there are multiple levels of skiplists (that is, if $c > 2$), the signature is verified recursively: that is, the verification of the signature requires the verification of a signature in a higher level chain. For example, if there are two levels of skiplists ($c = 3$), then the hash chain is followed to its

Table 1: Our mechanisms compared with public key equivalents

	Initialization Computation	Per-Hop Computation	Overhead (Bits)
Hash tree chain	$\mathrm{M}\cdot 120\mu s$	$(\mathrm{M}-\mathrm{m})\cdot 120\mu s$	1680
RSA Equivalent (CPU Optimized)	$.235\mu s$	$.235\mu s + \mathrm{m}\cdot 401\mu s$	$1024\mathrm{m} + 80\mathrm{m}\lg_2 \mathrm{N}\eta$
RSA Equivalent (Minimal Overhead)	$7669\mu s$	$\eta\cdot 7669\mu s + \mathrm{m}\cdot 401\mu s$	1040m
Tree Authenticated One-Way Chains	$1.5\mu s$	$3\mu s$	1600
RSA Equivalent	$7669\mu s$	$401\mu s$	1024
Skiplist	$(\mathrm{M}/\alpha)\cdot 120\mu s + \alpha\cdot step$	$(\mathrm{M}/\alpha)\cdot 120\mu s + 2\alpha\cdot step$	1920
RSA Equivalent	$(\mathrm{M}/\alpha)\cdot 7669\mu s + \alpha\cdot step$	$401\mu s + 2\alpha\cdot step$	$(\mathrm{M}/\alpha)\cdot 1024$

Notation: M is the maximum metric, m is the metric at a hop, n is the total network size, η is the average number of neighbors, α is the number of hops covered by one skiplist hop, and *step* is the cost of one hash chain step. RSA timings were performed with 1024-bit keys, using OpenSSL [29]. Hash tree chain performance is based on a network of size 32640, roughly the number of ASes in the Internet, and uses hash tree of size 2^6, with 3 values corresponding to each node. CPU optimized RSA equivalent combines all routing table elements using a Merkle tree, amortizing signature costs across all routing table elements. Tree-authenticated one-way chain is of size 2^{20}, and the calculation of initialization cost is amortized over all elements.

anchor, the second level skiplist signature is checked by following that skiplist to its anchor, and the anchor of that skiplist is verified by verifying the signature in the top-level skiplist.

Skiplists can be generalized to allow skipping over any type of one-way chain that is formed from a single arbitrary head and can be verified using a single anchor. For example, hash tree chains can be used in conjunction with skiplists. This generalized skiplist is generated in the same way as skiplists over hash chains: at the lowest level of skiplist, the head of one step is used to seed the head of the one-way chain, and the anchor of that one-way chain is signed by that step in the skiplist.

Another possible application of such skiplists is to choose the top-level skiplist to represent k steps, where k is the maximum diameter of the network. This would reduce the initial cost of setting up a Merkle tree (as described in Section 4.5) from $O(s\sqrt[c]{k})$ to $O(s)$, where s is the number of sequence numbers covered by the tree. This increases overhead and computation cost by $O(1)$ for each update sent and verified, respectively.

4.8. Efficiency Evaluation

To evaluate the efficiency of our mechanisms, we implemented generation and verification procedures for the three mechanisms described in this section. For efficiency, our hash function is based on the Rijndael block cipher [4] in the Matyas, Meyer, and Oseas construction [24], with a 128-bit key and a 128- or 192-bit block size, depending on the number of bits to be hashed. With a single block to be hashed, the hash output is the following (with an initialization vector (IV) as the initial key K): $H(x) = E_K(x) \oplus x$. We built our implementation on top of Gladman's implementation [9]. We implemented hash tree chains with 64 leaves, which represents a 64-node network with a single element per node, or a 2016-node network, when using two elements per node. Our skiplist was based on Rohatgi's construction [37] of 20 signature chains of length 16 and 3 checksum chains of length 16.

We ran our tests on a laptop with a Mobile Pentium 4 CPU running at 1.6GHz. Verifying a node in a tree-authenticated one-way chain took $3.08\mu s$ on average, computing one step in a hash tree chain took $120\mu s$ on average, and computing one step of an MW-chain took $145\mu s$ on average. As a result, in a network with maximum metric 16 using skiplists of length 4, the *worst case* verification takes just over one millisecond. Another advantage of our approach is that most of the computation needed for verification can be used for generation; in particular, the worst case authentication plus verification operation takes just $480\mu s$ more than verification alone.

To compare these results to the efficiency of public-key cryptography, we analyzed the functionality provided by each mechanism. A summary of our analysis is shown in Table 1. The tree-authenticated one-way chain essentially provides a signature: given a public key (the root value), private values can be authenticated. Tree-authenticated one-way chains are significantly more efficient than existing approaches [41] that authenticate each anchor using RSA.

A hash tree chain uses cryptographic mechanisms to ensure that only nodes authorized to advertise a particular metric can advertise that metric. In particular, only nodes that hear an advertisement with metric m (or lower) can advertise metric $m+1$. A public-key approach to this problem can be adapted from the solution proposed by Kent et al. [18]: each node signs the list of nodes that are allowed to advertise a particular metric. Each routing table element includes a signature chain, with a length equal to the metric, which shows the delegation of authority for advertising particular metrics. A node verifying this chain would need to verify a number of signatures equal to the distance to the destination. In addition, each node needs to

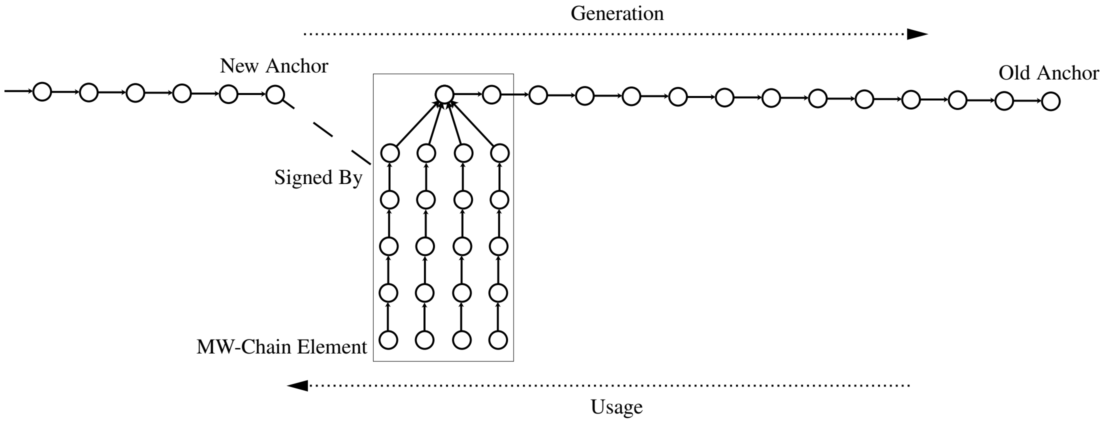

Figure 8: Bootstrapping a new hash chain. The new anchor is signed using the MW-chain element at the far left side of the old chain.

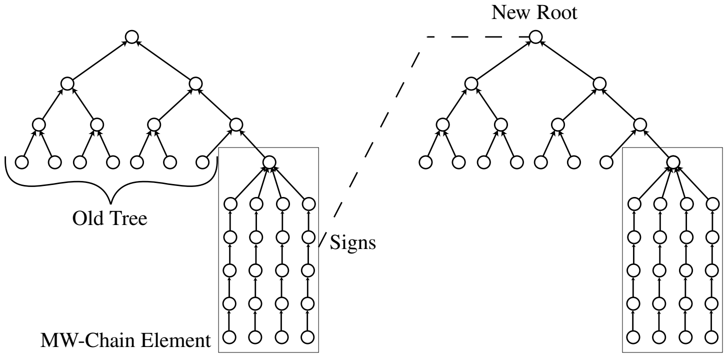

Figure 9: Bootstrapping a new tree-authenticated one-way chain. The new root is signed using the MW-chain element embedded in the old tree.

run a secure neighbor discovery protocol in order to know which neighbors to authorize. Though such a protocol may be easy to design in a wired network or a fixed wireless network, where a list of potential neighbors is easily generated, it could be prohibitively expensive in a mobile wireless environment such as an ad hoc network.

Finally, an alternative to skiplists is signatures. For example, in a network with maximum metric 16 and one step in a skiplist is used to skip over 4 elements, a sender can sign not only the anchor (metric 16 authenticator), it can also sign metric 4, 8, and 12 authenticators. Naturally, when a node sends an advertisement with metric 5, it will not include the signature of the metric 4 authenticator, and in general, a node advertising metric n will not include signatures on any metric m authenticators for $m < n$. Although skiplists may be slower than public key mechanisms on general-purpose processors, they have four advantages: first, they may require less network overhead for long chains; second, signature generation overhead is reduced, especially at the sender; third, they are easier to implement in hardware; and fourth, verification is easily parallelizable.

4.9. Bootstrapping New Chains and Trees

As time progresses, the elements of one-way hash chains or hash trees eventually run out and the node needs to securely distribute the anchor of a new chain or the root of a new tree. Recall that the security of our schemes relies on the secure distribution of these initial values, as they are used to authenticate all subsequent values. One solution to this problem is to compute a chain that is long enough to outlast the network; unfortunately, such a computation may be relatively expensive.

An alternative solution is to use the old chain or tree to authenticate the new anchor or root. To achieve this, we place a single MW-chain element at on the left-hand side of each hash chain or hash tree chain, or as the last element of

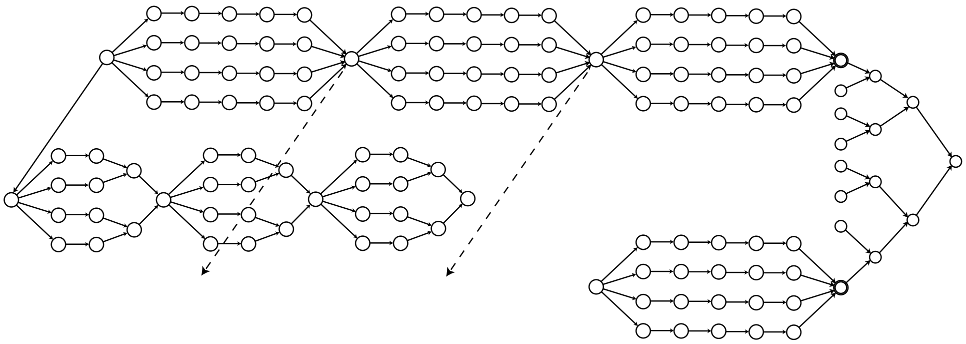

Figure 10: The Big Picture. Each leaf (except the bottom leaf) of the hash tree is identical to the top leaf, but the structure is omitted for clarity.

a tree-authenticated one-way chain (as used in Section 4.5). When a node comes close to running out of its current chain, it generates a new chain, and uses the MW-chain element from its old chain to form a one-time signature on the anchor of the newly generated chain. It then distributes this new anchor by piggybacking it on several updates around the time the old chain expires. Figure 8 shows the use of a skiplist element in a hash chain for authenticating a new hash chain anchor, and Figure 9 shows the use of a skiplist element in a tree-authenticated one-way chain for authenticating a new tree-authenticated one-way chain root.

This approach can be extended in several ways: each chain could contain several MW-chain elements to allow nodes to more easily reenter the network should they miss an entire chain of another node. Furthermore, a node may choose to piggyback a newly authenticated anchor often, when it first switches to the new chain, and progressively less often as the node consumes that chain. For example, the node may distribute the authentication information whenever the chain element used is a power of 2 from the anchor; this approach reduces overhead while still allowing nodes to rejoin the network after an extended time away.

4.10. Combining Our Primitives

Two of our primitives (hash tree chains described in Section 4.4 and tree-authenticated one-way chains in Section 4.5) protect against specific attacks (namely, same-metric fraud and the rushing attack); in addition, we provide skiplists (Section 4.6) for more efficient traversal of long hash chains. In order to prevent both of the above attacks, we can combine our approaches as shown in Figure 10. At the highest level, shown on the right side of the figure, we use a tree-authenticated one-way chain, which is a Merkle tree. The root of this Merkle tree is bootstrapped on each node. Each leaf in the Merkle tree is the anchor of another chain, with each leaf representing a single sequence number. In this case, the chains are skiplists built on top of hash tree chains; the chains could also be implemented as hash tree chains, or, if same-metric fraud is not a concern, as hash chains. Finally, at the bottom of the figure is an MW-chain element, which is later used for authenticating the root of the next tree-authenticated one-way chain.

5. A Mechanism for Securing Path Vector Protocols

5.1. Overview of Path Vector Routing

Path vector protocols are similar to distance vector protocols, except that in place of the metric, each routing update includes a list of routers (or, in the case of BGP, a list of Autonomous Systems) on the route. By default, a path vector protocol will choose a route with the shortest recorded path; policies may also specify specific routers to prefer or to avoid. As a result, a node may wish to authenticate each hop that the routing update has traversed as recorded in the path, and to assure that no hops were removed from that recorded path.

A traditional way to perform this authentication is to have each node insert an authenticator in the packet, and to have the recipient individually verify each authenticator when the packet is received. This approach requires the network overhead of carrying a message authentication code (MAC) for each node in the path. In this section, we present a *cumulative authentication* mechanism that has the property that the message can be authenticated with only a single MAC, together with an ordered list of nodes traversed by the packet.

5.2. Cumulative Authentication

First, we describe the cumulative authentication mechanism in the case in which private keys are shared between the authenticating node and each node on the path. Each packet authenticated in this way maintains a *path authenticator* and an *address list*. When the packet traverses a node, the node appends its address to the address list. It authenticates its position in the list by replacing the path authenticator with a MAC computed over the received path authenticator and the packet's immutable fields.

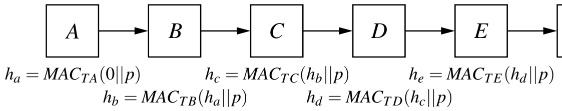

Figure 11: Cumulative authentication of packet p to a target T

When the packet reaches the receiver, if the path authenticator was originally initialized to a well known value (such as 0), then the receiver can reconstruct an expected final path authenticator value, given the address list. If the reconstructed value matches the received value, then the packet is deemed to be authentic and to have in fact traversed each node in the address list.

Figure 11 shows an example of cumulative authentication for a packet p. In addition to updating the path authenticator, each node also appends its own address to address list in the packet. If each node authenticates the packet using a shared MAC with T, then T can verify the path the packet traversed by verifying the received path authenticator h_e by checking that

$$h_e = MAC_{TE}(MAC_{TD}(MAC_{TC}(MAC_{TB}(MAC_{TA}(0 \mid\mid p) \\ \mid\mid p) \mid\mid p) \mid\mid p) \mid\mid p) \ .$$

Cumulative authentication also resists the removal of previous nodes from the address list. For example, in Figure 11, if an attacker C wishes to remove B from the address list, it must obtain h_a to derive a valid $h_c = MAC_{TC}(h_a||p)$. Since inverting B's MAC is infeasible, an attacker generally must have the cooperation of the node immediately before the node to be removed. This mechanism does not prevent the second node from removing the first node, but since the first node is the source node, this is equivalent to the second node dropping the original packet and originating a packet of the same type to the destination.

Instead of using private, shared keys for authentication, it is also possible to use our cumulative authentication mechanism in the case in which the TESLA broadcast authentication protocol [34, 35] is used for authentication; the authentication can be performed either by the sender of the packet to be authenticated or by each recipient. To perform the authentication at each recipient, as may be desirable with a proactive routing protocol, such as BGP, each node along the path verifies the TESLA "security condition" (that the TESLA keys have not yet been released) and updates a address list and path authenticator as described above using its current TESLA key. The node then buffers the packet for verification. Later, the sender transmits the key, required for the verification of its authentication, to each node to which the sender transmitted the original routing packet. Each node receiving such an authentication packet verifies the authentication information. After the node performs that authentication, it appends its previous TESLA key to the authentication packet and transmits the new authentication packet to each neighbor to which it sent the original routing packet.

In an on-demand protocol, such as Ariadne [14], an initiator floods a route request packet when it needs a route to a destination; the initiator may then wish to perform the authentication. In this case, each node along the path updates a address list and path authenticator as described above. When the packet reaches the destination, the destination verifies the TESLA security condition. Alternatively, the destination can include a timestamp, and allow the source to verify the security condition. The destination then adds an authenticator to the path authenticator and address list (and possibly the timestamp), and sends the packet along the reverse of the route along which it came. Each node receiving such a packet includes in the packet a key that allows the original authenticator to be reconstructed. If the end-to-end authentication is also performed using TESLA, the TESLA key used by the destination for authenticating the path authenticator, address list, and timestamp must be sent to the original sender.

5.3. Performance Evaluation

To evaluate the performance of cumulative authentication, we examined the overhead reduction resulting from using cumulative authentication together with Ariadne [14]. We performed 140 simulations, each running over 900 simulated seconds, and examined the number of bytes of overhead transmitted within control packets. When Ariadne was run without cumulative authentication, the *total* overhead across 50 nodes and 126000 simulated seconds was 1997 megabytes, whereas with cumulative authentication the same total overhead was 1491 megabytes. This result represents a 25% reduction in routing overhead.

6. Conclusion

In this paper, we have presented four new mechanisms as building blocks for creating secure distance vector and path vector routing protocols. These mechanisms not only protect the routing protocol against standard routing attacks, they are based on highly efficient *symmetric* cryptographic techniques; our mechanisms thus also help to protect the routing protocol against denial of service attacks based for example on simply by flooding large numbers of randomly generated, forged routing messages, which then must be authenticated and rejected by the routers.

For securing distance vector protocols, our *hash tree chain* mechanism forces a router to increase the distance (metric) when forwarding a routing table entry. To provide authentication of a received routing update in bounded time, we presented a new mechanism, similar to hash chains, that we call *tree-authenticated one-way chains*. For cases in which the maximum metric is large, we presented *skiplists*, which provide more efficient initial computation cost and more efficient element verification; this mechanism is based on a new cryptographic mechanism, called MW-chains, which we also presented. For securing path vector protocols, our *cumulative authentication* mechanism authenticates the list of routers on the path in a routing update, preventing removal or reordering of the router addresses in the list; this mechanism uses using only a single authenticator in the routing update rather than one per router address.

As our economy and critical infrastructure increasingly rely on the Internet, securing routing protocols becomes of critical importance. The routing security mechanisms we have described can be applied to conventional routing protocols such as those in use in the Internet today, as well as to specialized routing protocols designed for new environments such as multihop wireless ad hoc networking. Our mechanisms provide a foundation on which efficient secure routing protocols can be designed, and we leave the development of such protocols to future work.

7. Acknowledgments

We would like to thank Dawn Song for her important feedback on our work. We would also like to thank the anonymous reviewers for their valuable comments on an earlier draft of this paper and for encouraging us to include in the paper an evaluation of our proposed security mechanisms.

References

[1] Stefano Basagni, Kris Herrin, Emilia Rosti, and Danilo Bruschi. Secure Pebblenets. In *Proceedings of the Second Symposium on Mobile Ad Hoc Networking and Computing* (MobiHoc 2001), pages 156–163, October 2001.

[2] Steven Cheung. An Efficient Message Authentication Scheme for Link State Routing. In *Proceedings of the 13th Annual Computer Security Applications Conference*, pages 90–98, 1997.

[3] Don Coppersmith and Markus Jakobsson. Almost Optimal Hash Sequence Traversal. In *Proceedings of the Sixth International Conference on Financial Cryptography (FC 2002)*, Lecture Notes in Computer Science. Springer, 2002.

[4] Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael, March 1999.

[5] Bridget Dahill, Kimaya Sanzgiri, Brian Neil Levine, Elizabeth Royer, and Clay Shields. A Secure Routing Protocol for Ad hoc Networks. In *Proceedings of the 10th IEEE International Conference on Network Protocols* (ICNP '02), November 2002.

[6] Whitfield Diffie and Martin Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, November 1976.

[7] Shimon Even, Oded Goldreich, and Silvio Micali. On-Line/Off-Line Digital Signatures. In *Advances in Cryptology - Crypto '89*, edited by Gilles Brassard, volume 435 of *Lecture Notes in Computer Science*, pages 263–277. Springer-Verlag, 1989.

[8] Gregory Finn. Reducing the Vulnerability of Dynamic Computer Networks. Technical Report ISI-RR-88-201, USC/Information Sciences Institute, June 1988.

[9] Brian Gladman. Cryptography Technology: Implementations of AES (Rijndael) in C/C++ and Assembler, June 2002. Available at `http://fp.gladman.plus.com /cryptography_technology/rijndael/`.

[10] Ralf Hauser, Antoni Przygienda, and Gene Tsudik. Reducing the Cost of Security in Link State Routing. In *Proceedings of the 1997 Symposium on Network and Distributed Systems Security (NDSS '97)*, pages 93–99, February 1997.

[11] C. Hedrick. Routing Information Protocol. RFC 1058, June 1988.

[12] Andy Heffernan. Protection of BGP Sessions via the TCP MD5 Signature Option. RFC 2385, August 1998.

[13] Yih-Chun Hu, David B. Johnson, and Adrian Perrig. Secure Efficient Distance Vector Routing in Mobile Wireless Ad Hoc Networks. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA '02), pages 3–13, June 2002.

[14] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: A Secure On-Demand Routing Protocol for Wireless Ad Hoc Networks. In *Proceedings of the Eighth ACM International Conference on Mobile Computing and Networking* (MobiCom 2002), September 2002.

[15] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks. In *Proceedings of IEEE Infocomm 2003*, April 2003.

[16] Markus Jakobsson. Fractal Hash Sequence Representation and Traversal. In *Proceedings of the 2002 IEEE International Symposium on Information Theory (ISIT '02)*, pages 437–444, July 2002.

[17] John Jubin and Janet D. Tornow. The DARPA Packet Radio Network Protocols. *Proceedings of the IEEE*, 75(1):21–32, January 1987.

[18] Stephen Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure Border Gateway Protocol (S-BGP) — Real World Performance and Deployment Issues. In *Proceedings of the 2000 Symposium on Network and Distributed Systems Security (NDSS '00)*, pages 103–116, February 2000.

[19] Jiejun Konh, Petros Zerfos, Haiyun Luo, Songwu Lu, and Lixia Zhang. Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks. In *Proceedings of the Ninth International Conference on Network Protocols (ICNP '01)*, pages 251–260, November 2001.

[20] Brijesh Kumar. Integration of Security in Network Routing Protocols. *SIGSAC Review*, 11(2):18–25, 1993.

[21] Brijesh Kumar and Jon Crowcroft. Integrating Security in Inter Domain Routing Protocols. *Computer Communication Review*, 23(5):36–51, October 1993.

[22] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding BGP Misconfiguration. In *Proceedings of the SIGCOMM '02 Conference on Communications Architectures, Protocols and Applications*, August 2002.

[23] Gary Malkin. RIP Version 2. RFC 2453, November 1998.

[24] Stephen Matyas, Carl Meyer, and Jonathan Oseas. Generating Strong One-Way Functions with Cryptographic Algorithm. *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985.

[25] Ralph C. Merkle. Protocols for Public Key Cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, 1980.

[26] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology - Crypto '87*, edited by Carl Pomerance, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer-Verlag, 1987.

[27] Ralph C. Merkle. A Certified Digital Signature. In *Advances in Cryptology - Crypto '89*, edited by Gilles Brassard, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer-Verlag, 1989.

[28] Sandra Murphy. BGP Security Vulnerabilities Analysis. Internet-Draft, draft-murphy-bgp-vuln-01.txt, October 2002.

[29] OpenSSL Project team. OpenSSL, May 2000. `http://www.openssl.org/`.

[30] Charles E. Perkins and Pravin Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of the SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, August 1994.

[31] Charles E. Perkins and Elizabeth M. Royer. Ad-Hoc On-Demand Distance Vector Routing. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA'99), pages 90–100, February 1999.

[32] Radia Perlman. *Interconnections: Bridges and Routers*. Addison-Wesley, 1992.

[33] Adrian Perrig. The BiBa One-Time Signature and Broadcast Authentication Protocol. In *Proceedings of the Eighth ACM Conference on Computer and Communications Security (CCS-8)*, pages 28–37, November 2001.

[34] Adrian Perrig, Ran Canetti, Dawn Song, and J. D. Tygar. Efficient and Secure Source Authentication for Multicast. In *Proceedings of the 2001 Network and Distributed System Security Symposium, NDSS '01*, pages 35–46, February 2001.

[35] Adrian Perrig, Ran Canetti, J. D. Tygar, and Dawn Song. The TESLA Broadcast Authentication Protocol. *RSA CryptoBytes*, 5 (Summer), 2002.

[36] Yakov Rekhter and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, March 1995.

[37] Pankaj Rohatgi. A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, November 1999.

[38] Karen E. Sirois and Stephen T. Kent. Securing the Nimrod Routing Architecture. In *Proceedings of the 1997 Symposium on Network and Distributed Systems Security (NDSS '97)*, February 1997.

[39] Bradley R. Smith and J.J. Garcia-Luna-Aceves. Securing the Border Gateway Routing Protocol. In *Proceedings of Global Internet'96*, pages 81–85, November 1996.

[40] Bradley R. Smith, Shree Murthy, and J.J. Garcia-Luna-Aceves. Securing Distance Vector Routing Protocols. In *Proceedings of the 1997 Symposium on Network and Distributed Systems Security (NDSS '97)*, pages 85–92, February 1997.

[41] Manel Guerrero Zapata and N. Asokan. Securing Ad Hoc Routing Protocols. In *Proceedings of the ACM Workshop on Wireless Security* (WiSe 2002), September 2002.

[42] Kan Zhang. Efficient Protocols for Signing Routing Messages. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS '98)*, March 1998.

[43] Lidong Zhou and Zygmunt J. Haas. Securing Ad Hoc Networks. *IEEE Network Magazine*, 13(6):24–30, November/December 1999.