Leap-Frog Packet Linking and Diverse Key Distributions for Improved Integrity in Network Broadcasts

MICHAEL T. GOODRICH
Dept. of Computer Science
University of California, Irvine
Irvine, CA 92697-3425
goodrich (at) ieee.org

Abstract

We present two new approaches to improving the integrity of network broadcasts and multicasts with low storage and computation overhead. The first approach is a leap-frog linking *protocol for securing the integrity of packets as they traverse a network during a broadcast, such as in the setup phase for link-state routing. This technique allows each router to gain confidence about the integrity of a packet before passing it on to the next router; hence, allows many integrity violations to be stopped immediately in their tracks. The second approach is a novel key predistribution scheme that we use in conjunction with a small number of hashed message authentication codes (HMACs), which allows end-to-end integrity checking as well as improved hop-by-hop integrity checking. Our schemes are suited to environments, such as in ad hoc and overlay networks, where routers can share only a small number of symmetric keys. Moreover, our protocols do not use encryption (which, of course, can be added as an optional security enhancement). Instead, security is based strictly on the use of one-way hash functions; hence, our algorithms are considerably faster than those based on traditional public-key signature schemes. This improvement in speed comes with only modest reductions in the security for broadcasting, as our schemes can tolerate small numbers of malicious routers, provided they don't form significant cooperating coalitions.*

1 Introduction

The integrity of messages in network broadcasts and multicasts is an essential component of communication, as false or modified packets contribute to congestion and network overhead. Moreover, falsified packets can be used in denial-of-service attacks or network infrastructure attacks. In ad hoc networks, for example, manufactured false broadcasts can be used to run down the batteries of devices being used as routers. In addition, the network infrastructure itself is vulnerable to falsified broadcasts, as the algorithms that form the basis of most routing protocols, such as OSPF and BGP, use broadcasts as a primitive (e.g., see [15, 25]). Indeed, some of these algorithms have been compromised by routers that did not follow the respective protocols correctly. Fortunately, network malfunctions resulting from faulty routers have to date been shown to be the result of misconfigured routers, not malicious attacks. Nevertheless, these failures show the feasibility of malicious broadcast or multicast attacks, for they demonstrate that compromising a single router can undermine the performance of an entire network.

We are therefore interested in studying ways of improving the integrity of packets in network broadcasts and multicasts where routers can store a small number of keys and can perform a limited number of computations using those keys. Here we use the term "router" fairly loosely to refer to any device that receives and forwards packets in a network broadcast or multicast, even if this routing takes place at the application layer. Such a network could be an autonomous system on the Internet, an ad hoc network, or an overlay network used for multicasting or peer-to-peer applications. In many of these applications, the routers are potentially low-computation devices or have limited computing resources that they can devote to routing packets. Thus, we desire solutions that are efficient. Indeed, we would like to limit the security computations that routers make for achieving integrity to use only the fastest of cryptographic primitives, such as one-way hash functions. We are not explicitly requiring that we also achieve confidentiality for the broadcast messages, however, since in many applications integrity is more important than confidentiality.

1.1 Prior Related Work

Network broadcast security was first studied in the seminal work of Perlman [24] (see also [25]), who studied flooding algorithms that are resilient to faulty routers. Her schemes are based on using a public-key infrastructure where each router x is given a public-key/private-key pair and must sign each message that originates from x. Likewise, in her schemes, any router y that wants to authenticate a message M checks the signature of the router x that originated it. Such a signature-based approach is sufficient, therefore, to achieve integrity in a broadcast or multicast algorithm. Even so, several researchers have commented that, from a practical point of view, requiring full public-key signatures on all broadcast messages is probably not efficient, particularly for environments where routers are low-computation devices. Signing and checking signatures are expensive operations when compared to the simple table lookups and computations performed in the well-known routing algorithms. Nevertheless, there has been considerable previous work discussing the details of protocols that would implement integrity through the use of digital signatures, including work by Guerrero-Zapata and Asokan [11], Kent *et al.* [16], Konh *et al.* [17], Murphy *et al.* [20, 22, 21], Papadimitratos and Haas [23], Sanzgiri *et al.* [27], and Smith *et al.* [29].

Motivated by the desire to create efficient and secure broadcast or multicast algorithms, several researchers have designed algorithms that achieve security at computational costs that are argued to be superior to those of Perlman. Given that the signature-based design of Perlman is already highly-secure, this research has used fast cryptographic tools, such as one-way hash functions, instead of public-key digital signatures on all messages. Nevertheless, since there is a natural trade-off between computational speed and security, this research has also involved the introduction of additional assumptions about the network or restrictions on the kinds of network attacks that one is likely to encounter. The challenge, then, is to create practical and secure broadcast algorithms using fast cryptographic tools while limiting the security assumptions needed for these algorithms to maintain packet integrity.

Cheung [4] shows how to use hash chaining to secure broadcast algorithms, assuming that the routers have synchronized clocks. His scheme is not timely, however, as it can only detect attacks long after they have happened. Hauser, Przygienda, and Tsudik [12] avoid that defect by using hash chains to instead reveal the status of specific links in a link-state algorithm. That is, their protocol is limited to simple yes-no types of messages. In addition, because of the use of hash chains, they also require that the routers in the network be synchronized. Zhang [33] extends their protocol for more complex messages, but does so at the expense of many more hash chains, and his protocol still requires synchronized routers. It is not clear, in fact, whether his scheme would actually be faster than a full-blown digital signature approach, as advocated in the early work of Perlman. Also of related interest, is work of Bradley *et al.* [2], who discuss ways of improving the security of packet delivery after the routing tables have been built. In addition, Wu *et al.* [32] and Vetter *et al.* [31] discuss some practical and empirical issues in securing routing algorithms.

Recently, Hu, Perrig, and Johnson [13] show how to use chains of one-way hash functions to improve the integrity of the setup packets used to build routing tables for distance vector and path vector routing. Likewise, Zhu *et al.* [35] show how to use one-way hash chains for hop-by-hop authentication. Our first approach complements these recent works, in that we use small sets of one-way hash functions to improve the integrity of packets as they are being used for broadcasts.

Since our second scheme involves the use of a novel randomized key pre-distribution method, previous work on randomized key pre-distribution is also relevant to the topics of this paper. Eschenauer and Gligor [8] propose a randomized key pre-distribution scheme based on creating a large pool of potential keys and having each device (or router) select a random subset of this pool as its keys. These keys are used for point-to-point unicast routing by having a sender use a key known to be shared by the receiver, for encryption or integrity. Chan *et al.* [3] improve and analyze several of the features of the Eschenauer-Gligor approach for unicast routing algorithms, keeping to the basic framework of using a single key pool. Zhu *et al.* [34] show how improve the key identification computation for these unicast routing schemes by using a pseudo-random number generator seeded with each node's ID to choose the keys from the key pool. Du *et al.* [7] show how to combine the Eschenauer-Gligor scheme and a pairwise key-generation scheme of Blom [1] to allow unicast routing with guaranteed shared keys between sender-receiver pairs, using less memory. Likewise, Liu and Ning [18] use a pooled polynomial-based key distribution scheme to achieve similar results. Hwang and Kim [14] study the connectivity properties of these key-distribution schemes for establishing pairwise secure connections. All of these schemes are effective for unicast routing, but they are not directly applicable for efficient broadcast or multicast routing.

1.2 Our Results

In this paper we describe two new approaches to improving the data integrity of broadcast and multicast algorithms on devices with low storage and computational resources. After a preliminary setup that involves distributing a set of small set of secret keys to the routers, our schemes use

simple cryptographic hashed message authentication codes (HMACs) to achieve security.

Our first approach involves the use of a technique we call *leap-frog* linking between hops of a packet as it is routed, for it allows parties in a broadcast tree to authenticate messages between every other member in a path from the source. This scheme achieves data integrity using hashed message authentication codes (HMACs) in broadcast messages under the assumption that there are no two adjacent malicious routers that are colluding with each other. Such a strategy would even be effective, for example, for broadcasts and multicasts in peer-to-peer networks, which are notoriously insecure but are likely to experience few collusion attacks. Our algorithms allow a router to receive messages from an untrusted neighbor in such a way that the neighbor cannot modify the message contents without being detected. The number of keys used per router in this scheme is at most its network degree, which is the minimum storage need per device just to route messages.

The second approach is a *diverse* key distribution scheme that uses a small number of keys per device and HMACs to achieve end-to-end integrity checking as well as improved hop-by-hop integrity checking for network broadcasts and multicasts. The main idea of this approach is to distribute L keys to each router in a diverse way, so that the intersection of the sets of keys between two routers is neither too small nor too big (we make this notion more formal in the paper). This approach differs from previous randomized key pre-distribution schemes [3, 7, 8], in that it is based on the use of L sets of *colored* keys, with devices picking one color from each set, rather than a large pool of similar keys from which devices choose a subset. We show that our distribution scheme can in fact be done using only a logarithmic or fewer number of keys. Such a distribution allows a filtering scheme, whereby routers can be confident of the data integrity of packets, subject to the (stronger) assumption that at most $\Omega(\log L)$ malicious routers are colluding with each other to spoof this filtering scheme. It also guarantees that even though we are using only a small number of keys for authentication, every node in a broadcast or multicast will be guaranteed to be able to share a key with the sender (in fact, they will share several keys).

Our protocols do not use encryption (which, of course, can be added as an optional security enhancement); the only cryptographic primitive utilized is the use of one-way hash functions in hashed message authentication codes (HMACs). This usage allows our algorithms to be considerably faster than those based on traditional public-key signature schemes.

2 Leap-Frog Packet Linking

We begin by discussing a low-cost way of making broadcast flooding and multicast routing more secure. Our method involves the use of a novel "leap frog" message-authenticating scheme.

2.1 The Network Framework

Let $G = (V, E)$ be a network whose vertices in V are considered as routers and whose edges in E are connections between these routers. We assume that the routers have some convenient addressing mechanism that allows us without loss of generality to assume that the routers are numbered 1 to n. We assume the network allows for the routing or flooding of messages. We also assume (for the basic leap-frog scheme) that the network topology is static.

For completeness, let us briefly review the broadcast flooding algorithm, so that we can identify how data integrity plays an important role in its correctness. The flooding algorithm is initiated by some router s creating a message M that it wishes to send to every other router in G. The typical way the flooding algorithm is implemented is that s incrementally assigns sequence numbers to the messages it sends. So that if the previous message that s sent had sequence number j, then the message M is sent with sequence number $j + 1$ and an identification of the message source, that is, as the message $(s, j+1, M)$. Likewise, every router x in G maintains a cache S_x that stores the largest sequence number encountered so far from each recently-encountered broadcast source router in G. Thus, any time a router x receives a message $(s, j+1, M)$ from an adjacent router y the router x first checks if $S_x[s] < j + 1$. If so, then x assigns $S_x[s] = j + 1$ and x sends the message $(s, j + 1, M)$ to all of its adjacent routers, except for y. If the test fails, however, then x assumes it has handled this message before and it discards the message.

If all routers perform their respective tasks correctly, then the flooding algorithm will send the message M to all the nodes in G. Indeed, if the communication steps are synchronized and done in parallel, then the message M propagates out from s in a breadth-first fashion.

If the security of one or more routers is compromised, however, then the flooding algorithm can be successfully attacked. For example, a router t could spoof the router s and send its own message $(s, j + 1, M')$. If this message reaches a router x before the correct message, then x will propagate this imposter message and throw away the correct one when it finally arrives. Likewise, a corrupted router can modify the message itself, the source identification, and/or the sequence number of the full message in transit. Each such modification has its own obvious bad effects on the network. For example, incrementing the sequence number

to $j+m$ for some large number m will effectively block the next m messages from s. Indeed, such failures have been documented (e.g., [32, 31]), although many such failures can be considered router misconfigurations not malicious intent. Of course, from the standpoint of the source router s the effect is the same independent of any malicious intent—all flooding attempts will fail until s completes m attempted flooding messages or s sends a sequence number reset command (but note that the existence of unauthenticated reset commands itself presents the possibility for abuse).

2.2 The Basic Leap-Frog Protocol

One possible way of avoiding the possible failures that compromised or misconfigured routers can inflict on a flooding algorithm is to take advantage of a public-key infrastructure defined for the routers. In this case, we would have s digitally sign every flooding message it transmits, and have every router authenticate a message before sending it on [24, 25]. Unfortunately, this approach is more computationally expensive than a scheme based instead on cryptographic hashing. For example, benchmarking tests (e.g., see [6, 26]) support the working assumption that cryptographic hash functions are 5,000 to 10,000 times faster than most public-key signature verification algorithms and 500 to 1,000 times faster than RSA signature verification with a simple public exponent (such as $2^{16}+1$).

Our scheme is based on a light-weight cryptographic hashing strategy, which we call *leap-frog* linking. The initial setup for our scheme involves a simple key distribution. Specifically, we define for each router x the set $N(x)$, which contains the vertices (routers) in G that are neighbors of x (which does not include the vertex x itself). That is,

$$N(x) = \{y\colon (x,y) \in E \text{ and } y \neq x\}.$$

The security of our scheme is derived from a secret key $k(x)$ that is shared by all the vertices in $N(x)$, but not by x itself. These keys can be created in a setup phase, when the routing devices are first deployed, or can be maintained by a network administrator. Note, in addition, that $y \in N(x)$ if and only if $y \in N(y)$.

Now, when s wishes to send the message M as a flooding message to a neighboring router, x, it sends

$$(s, j+1, M, h(s||j+1||M||k(x)), 0),$$

where h is a cryptographic hash function that is collision resistant (e.g., see [28]). Any router x adjacent to s in G can immediately verify the authenticity of this message (except for the value of this application of h), for this message is coming to x along the direct connection from s. But nodes at distances greater than 1 from s cannot authenticate this message so easily when it is coming from a router other than s. Fortunately, the propagation protocol will allow for all of these routers to authenticate the message from s, under the assumption that at the malicious routers along routing paths do not collaborate during the computation.

Let $(s, j+1, M, h_1, h_2)$ be the message that is received by a router x on its link from a router y. If $y = s$, then x is directly connected to s, and $h_2 = 0$. But in this case x can directly authenticate the message, since it came directly from s. In general, for a router x that just received this message from a neighbor y with $y \neq s$, we inductively assume that h_2 is the hash value $h(s||j+1||M||k(y))$. Since x is in $N(y)$, it shares the key $k(y)$ with y's other neighbors; hence, x can authenticate the message from y by using h_2. This authentication is sufficient to guarantee correctness, assuming no more than one router is corrupted at present, even though x has no way of verifying the value of h_1. So to continue the propagation assuming that flooding should continue from x, the router x sends out, to the next router w on the path (in the case of a unicast) or each w that is x's neighbor (in the flooding case), the message

$$(s, j+1, M, h(M||j+1||k(w)), h_1).$$

Note that this message is in the correct format for each such w, for h_1 should be the hash value $h(s||j+1||M||k(x))$, which w can immediately verify, since it knows $k(x)$. Note further that, just in the flooding case, the first time a router w receives this message, it can process it, updating the sequence number for s and so on. (See Figure 1.)

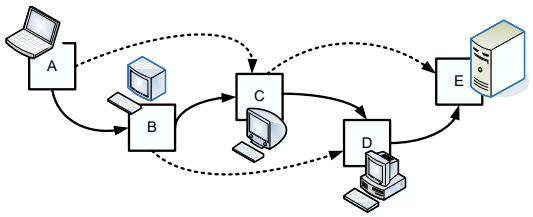

Figure 1. Illustrating leap-frog packet linking. We show the hops in the route from A to E using solid lines. The leap-frog linking in the integrity validation is shown using dotted lines.

This simple protocol has a number of performance advantages. First, from a security standpoint, inverting or finding collisions for a cryptographic hash function is computationally difficult. Thus, it is infeasible for a router to fake a hash authentication value without knowing the shared key of its neighbors, should it attempt to alter the contents of the message M.

Another advantage of this protocol is its computational efficiency. The only additional work needed for a router x to complete its processing for a flooding message is for

x to perform one hash computation for each of the edges of G that are incident on x. That is, x need only perform $\mathrm{degree}(x)$ hash computations, where $\mathrm{degree}(x)$ denotes the degree of x. Likewise, x need only store $\mathrm{degree}(x)$ keys in order to perform this protocol (which is the minimum storage needed just to forward broadcast flooding messages). Typically, for communication networks, the degree of a router is kept bounded by a constant. Thus, this work and storage compares quite favorably in practice to what would be required to verify a full-blown digital signature from a message's source.

The leap-frog routing process can detect a router malfunction in the flooding algorithm, for any router y that does not follow the protocol will be discovered by one of its neighbors x. Assuming that x and y do not collude to suppress the discovery of y's mistake in this case, then x can report to s or even a network administrator that something is potentially wrong with y. For in this case, y has clearly not followed the protocol. In addition, note that this discovery will occur in just one message hop from y.

2.3 Chromatic Leap-Frog Packet Linking

In some contexts it might be too expensive for a router to perform as many hash computations as it has neighbors in the case of a broadcast flooding message. Thus, we might wonder whether it is possible to reduce the number of hashes that an intermediate router needs to do to one even for flooding messages. In this subsection we describe how to achieve such a result, albeit at the expense of increasing the size of the message that is sent (but still keeping the storage per device to be at most equal to its network degree). Since our method is based on a coloring of the vertices of G, we refer to this scheme as the *chromatic leap-frog* approach.

In this scheme, we change the preprocessing step to that of computing a small-sized coloring of the vertices in G so that no two adjacent nodes are assigned the same color. Algorithms for computing or approximating such colorings are known for a wide variety of graphs. For example, every tree can be colored with two colors. Such colorings might prove useful in applying our scheme to multicasting algorithms, since multicasting communications often take place in trees. In addition, every planar graph can be colored with four colors with some difficulty and easily with five. Such graphs could arise naturally from distributed sensor networks. Finally, it is easy to color a graph that has maximum degree d using at most $d + 1$ colors by a straightforward greedy algorithm. This last class of graphs is perhaps the most important for general networking applications, as most communications networks bound their degree by a constant.

Let the set of colors used to color G be simply numbered from 1 to c and let us denote with V_i the set of vertices in G that are given color i, for $i = 1, 2, \ldots, c$, with $c \geq 2$. As a preprocessing step, we create a secret key k_i for the color i. We do not share this color with the members of V_i, however. Instead, we share k_i with all the vertices that are *not* assigned color i.

When a router s wishes to route or flood a message M with a new sequence number $j + 1$, in this new secure scheme, it creates a full message as

$$(s, j + 1, M, h_1, h_2, \ldots, h_c),$$

where each $h_i = h(s||j + 1||M||k_i)$. There is one problem for s to build this message, however. It does not know the value of k_i, where i is the color for s. So, it will set that hash value to 0. Then, s sends this message to each of its neighbors.

Suppose now that a router x receives a message of the form

$$(s, j + 1, M, h_1, h_2, \ldots, h_c)$$

from its neighbor s. In this case x can verify the authenticity of the message immediately, since it is coming along the direct link from s. Thus, in this case, x does not need to perform any hash computations to validate the message. Still, there is one hash entry that is missing in this message (and is currently set to zero): namely, $h_i = 0$, where i is the color of s. In this case, the router x computes $h_j = h(s||j + 1||M||k_j)$, since it must necessarily share the value of k_j, by the definition of a vertex coloring. The router x then sends out the (revised) message $(s, j + 1, M, h_1, h_2, \ldots, h_c)$.

Suppose then that a router x receives a message $(s, j + 1, M, h_1, h_2, \ldots, h_c)$ from its neighbor $y \neq s$. In this case we can inductively assume that each of the h_i values is defined. Moreover, x can verify this message by testing if $h_i = h(s||j + 1||M||k_i)$, where i is the color for y. If this test succeeds, then x accepts the message as valid and sends it on to all of its neighbors except y, to continue the broadcast. In this scheme, the message is easily authenticated, since y could not manufacture the value of h_i.

If a router modifies the contents of M, the identity of s, or the value of $j + 1$, this alteration will be discovered in one hop. Nevertheless, we cannot immediately implicate a router x if its neighbor y discovers an invalid h_i value, where i is the color of x. The reason is that another router, w, earlier in the flooding could have simply modified this h_i value, without changing s, $j + 1$, or M. Such a modification will of course be discovered by y, but y cannot know which previous router performed such a modification. Thus, we can detect modifications to content in one hop, but we cannot necessarily detect modifications to h_i values in one hop. Even so, if there is at most one corrupted router in G, then we will discover a message modification if it occurs. If the actual identification of a corrupted router is important

for a particular application, however, then it might be better to use the non-chromatic leap-frog scheme, since it catches and identifies a corrupted router in one hop.

2.4 Dealing with Network Updates

Since networks are rarely static, it is natural to address the computations that are needed for leap-frog linking to deal with network updates. Since there is no revocation mechanism in our scheme, deleting nodes and edges from the network requires no changes.

Inserting new nodes and edges with respect to the basic leap-frog linking scheme requires some work, however. Adding a new node x, with neighbor set $N(x)$, to the network requires that the administrator compute a new key $k(x)$ and distribute it to all the nodes in $N(x)$. Likewise, adding a new edge (x, y) to the network requires that the administrator inform x of $k(y)$ and inform y of $k(x)$. These communications are assumed to be done out of band (or using encryption).

Inserting a new node x with respect to chromatic leap-frog linking is more efficient than in the basic scheme. In this case, we assign x a color i that is different from all of x's neighbors and we communicate to x all the of color keys except for the key k_i.

Adding a new edge (x, y) in the chromatic scheme is potentially more problematic. If x and y are colored differently, then there is nothing to do, with respect to the keys stored at x and y. The previously-distributed color keys will still work. But if x and y are currently the same color, then we need to recolor the graph and distribute new color keys based on this new coloring.

Thus, the leap-frog schemes are best suited to contexts where the network topology is fairly static. Incidentally, our scheme based on diverse key distributions, which we describe in Section 3, is more tolerant of arbitrary network topology changes.

2.5 Evaluation and Analysis

The principle advantage of the leap-frog scheme is that allows for immediate integrity checking, without waiting for the future revelation of the pre-image of a one-way hash function (as in the previous schemes based on hash chains [13, 35]). Thus, comparing with a previous solution for immediate integrity checking, we compare our solutions with the public-key signature scheme of Perlman [24] (see also [25]). In either case, whether we are using an HMAC or digital signature to authenticate a message, we are most likely going to be first producing a digest of the message using a cryptographic hash function. Table 1 shows estimates, based on the Crypto++ 5.2.1 Benchmarks [6], of the time needed to construct such a digest, depending on the size of the message M.

Alg.	10 B	100 B	1 KB	10 KB	100 KB
MD5	.046	.46	4.6	46	460
SHA1	.147	1.47	14.7	147	1,470

Table 1. Running times, in microseconds, for computing a digest of a message of various sizes, based on the Crypto++ 5.2.1 Benchmarks [6].

Moreover, since the prime alternative to the leap-frog scheme is full digital signatures, we show in Table 2 the benchmark times for digital signatures.

Alg.	Sign	Verify
RSA 1024	4,750	180
RSA 2048	28,130	450
DSA 1024	2,180	2,490

Table 2. Digital signature computation and verification times, in microseconds, based on the Crypto++ 5.2.1 Benchmarks [6], for a digested message.

Efficiency. In the standard leap-frog scheme, a router needs to perform a number of hash computations equal to its degree in order to forward a broadcast message to its neighbors. That is, a router x processing a broadcast performs d cryptographic hashes, where d is the number of x's neighbors in the network. Using the heuristic that computing a cryptographic hash function is $1,000$ times faster than a digital signature check, we can conclude for the additional time required for authenticating a digest, that leap-frog integrity checking is faster than digital signature checking whenever the degree of routers in the network is less than $1,000$, which should be the case in most instances. Being more specific, each of the d hashes must be performed on a string of roughly 50 bytes. Table 3 shows the estimated time needed to perfom these hashes as a function of d, the degree of the router. The setup for performing the basic leap-frog scheme is just a single hash, of course, which is benchmarked as .23 microseconds for MD5 or .74 microseconds for SHA1 [6].

The additional time for the routing step in the chromatic version of leap-frog integrity checking will always be faster than digital signature checking, of course, since each router need perform only one hash computations per

	5	10	20	50	100	200
MD5	1.15	2.3	4.6	11.5	23	46
SHA1	3.7	7.4	14.7	36.8	73.5	147

Table 3. Times for performing d hashes on 50 B data, in microseconds, based on the Crypto++ 5.2.1 Benchmarks [6].

broadcast message—either to authenticate the message using the color key of the sending router (which that router doesn't know) or one to produce an HMAC with the color key of the sending router (in the direct connection to the sender case). That is, the verfication step for the chromatic leap-frog scheme involves computing a single hash, which is benchmarked as .23 microseconds for MD5 or .74 microseconds for SHA1 [6]. The setup step for the chromatic leap-frog scheme requires $d-1$ hashes, where d is the number of colors used in our scheme. Thus, we can use Table 3 to estimate the additional setup time for performing a broadcast in the chromatic leap-frog scheme.

Like standard digital signatures, the leap-frog scheme requires a non-trivial static key pre-distribution to the routers, namely each of the keys for each neighbor set. Such a distribution scheme might be appropriate for a LAN or even a set of wireless base stations. Thus, leap-frog checking would be an efficient means to achieve integrity in network broadcasts. But leap-frog checking is not an efficient solution in dynamic networks, including peer-to-peer and ad hoc networks, where routers can be added to the network dynamically. For such dynamic scenarios, the integrity checking scheme we describe in Section 3 would be a better choice.

Security. We claim that our leap-frog schemes can detect the existence of a malicious router that attempts to modify a broadcast message from a different sender or that attempts to inject a spoofed message with a source ID other than itself. This claim can of course be extended to multiple malicious routers, assuming that they do not collude (that is, a malicious router is willing to implicate a malicious router other than itself). Of course, if the network is not biconnected and a malicious router is an articulation point, then it can drop messages without being detected. So let us assume that the network is biconnected (which is usually the case in practice).

Assuming that a router x has no knowledge of the key $k(x)$ shared by x's neighbors (or the key k_i corresponding to x's color), the message sent by x contains a keyed HMAC that uses a key unknown to x, but known to the predecessor and successor of x on this path. Thus, without inverting a cryptographic hash function, if x modifies a broadcast message or if x attempts to send a spoofed message, it will be caught.

Of course, if two malicious routers x and y are adjacent and colluding, then x can change a message and compute an HMAC for it using $k(y)$. If y is then willing to compute an HMAC for this changed message using a key $k(z)$ and send this to a third router z, then z will accept the false message. Indeed, if the colluding and adjacent routers x and y simply report their respective neighbor keys $k(x)$ and $k(y)$ to the other, then either x or y can create a falsified message without any additional help from the other router. So we must assume in the leap-frog scheme that malicious routers do not collude. This is likely a reasonable assumption in practice, and we should not be surprised that this scheme has reduced security over a scheme based on public-key digital signature verification, which would be many times slower.

3 Diverse Key Distribution

Let us now discuss the technique of *diverse* key distribution. As mentioned above, the main idea of this technique is to distribute a small number of keys to each router so that every pair of routers shares a set keys, but it would take a considerable number of routers to collude to cover all of these keys.

3.1 Achieving Diversity Through Overlapping and Uncensorable Key Sets

Let us begin with a few definitions. Let $\mathcal{K}$ be a set of keys that are to be distributed to a set S of n devices so that each device i in S will store a set $\mathcal{K}_i$ of L keys from $\mathcal{K}$. We say that such a key distribution is d-*overlapping* if the number of keys shared by any two devices is at least d, that is,

$$|\mathcal{K}_i \cap \mathcal{K}_j| \geq d,$$

for $i \neq j$. We define such a key distribution to be g-*uncensorable*[1] if, for any two devices i and j, the number of other devices needed to cover all of the keys in the intersection of i and j's key sets is at least g, that is, we need at least g sets, $\mathcal{K}_{i_1}, \mathcal{K}_{i_2}, \ldots, \mathcal{K}_{i_g}$ so that

$$\mathcal{K}_i \cap \mathcal{K}_j \subseteq \mathcal{K}_{i_1} \cup \mathcal{K}_{i_2} \cup \cdots \cup \mathcal{K}_{i_g}.$$

A key distribution $\{\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n\}$ is (d, g)-*diverse* if it is d-overlapping and g-uncensorable. The goal, of course, is to construct a (d, g)-diverse key distribution of small size, L, but with d and g being as large as possible. Before we describe a way of constructing such a key distribution, let us describe how we could use it to achieve improved integrity in network routing.

[1] Our definition of a g-uncensorable key distribution is equivalent to a $(2, g-1)$-cover-free set system, using the terminology of Stinson *et al.* [30].

3.2 Broadcasting among Devices Having a Diverse Key Distribution

A (d, g)-diverse key distribution allows us to have a rich and robust set of keys to use in HMACs for any message that is to be broadcast in our network. For example, suppose a device, i, wants to broadcast a message M. The device i can simply use all of the keys in its $\mathcal{K}_i$ set, creating an HMAC for each one (this requires L calls to a one-way hash function). A device j receiving the message M and its HMACs can then be assured that it has at least d keys in its $\mathcal{K}_j$ set that it shares with i; hence, these keys can be used to validate at least d of the HMACs that i sent (this takes between d and L calls to a one-way hash function). Moreover, should all these HMACs (using keys in $\mathcal{K}_i \cap \mathcal{K}_j$) turn out to be valid, then the device j can place considerable trust in the integrity of the message M, for it would take at least g of the routers on the path from i to j to collude in order to change M to some alternate M' in way that could still be validated with all of the keys in $\mathcal{K}_i \cap \mathcal{K}_j$. (See Figure 2.)

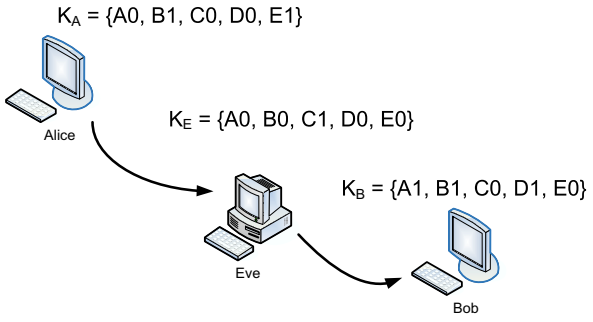

Figure 2. An illustration of using HMACs with a diverse key distribution. Notice that the keys B1 and C0 are shared between Alice and Bob but not Eve.

3.3 Achieving Greater Integrity Through Intermediate Validations

The above scheme has the nice property that honest intermediate routers, sitting between the devices i and j, need do no additional work for the sake of validation. They just need to forward the packets from i to j. We can optionally add intermediate validation to our scheme, however, with modest overhead. This would allow us, for example, to stop falsified packets supposedly being sent from i to j long before they reach j. The idea would be to have each intermediate router m check the validity of the message coming from i using the keys in the set $\mathcal{K}_i \cap \mathcal{K}_m$ (or even just a small random sample from this set, since this validation will be repeated by the other honest routers on the path from i to j). If the device m discovers that the message has been compromised, then device m can simply discard the packet, saving device j (and the other downstream routers) the trouble.

3.4 Constructing a Diverse Key Distribution

Of course, the efficiency of using HMACs with a diverse key distribution depends on our being able to create such a distribution using a small universe, $\mathcal{K}$, of keys. That is, we would like L to be small, while allowing for d and g to be relatively large. Fortunately, we can create such a key distribution without too much overhead, for we show in this section that can distribute to each router i a suitably-chosen random set $\mathcal{K}_i$ of L keys, taken from a universe $\mathcal{K}$ of $2L$ keys, in way that is likely to be (d, g)-diverse, where L is $\Theta(\log n)$, d is $\Omega(\log n)$, and g is $\Omega(\log L)$.

Such a distribution would go as follows. We begin by setting L as a security parameter, but keeping L to be $O(\log n)$, where n is the number of routers. For example, as we show later in this section, we could choose L to be 20 to achieve a 97% likelihood of detecting any falsified packet, or we could set L to be $8\lceil \log n \rceil$ to achieve a near certain probability of detecting any falsified packet. Given the security parameter L, we create $\mathcal{K}$ to be a set of $2L$ randomly-chosen keys. This will be our key universe.

We pair up the keys in $\mathcal{K}$ into L pairs, with one member of each pair being viewed as a "0-bit" key and the other being viewed as a "1-bit" key. We number these pairs $1, 2, \ldots, L$. We then assign to each device i a random string of L bits, $b_1 b_2 \ldots b_L$, and we build the set $\mathcal{K}_i$ by selecting the keys from $\mathcal{K}$ that correspond to the bits in this string. That is, if $b_l = 0$, then we include in $\mathcal{K}_i$ the l-th 0-bit key; otherwise, if $b_l = 1$, then we include in $\mathcal{K}_i$ the l-th 1-bit key. This simple random selection process is related to the randomized bucketed key assignment scheme of Garay *et al.* [9] and is likely to give us a diverse key distribution.

Theorem 1: *For any pair of devices i and j, $|\mathcal{K}_i \cap \mathcal{K}_j| \geq L/8$, with high probability. The probability this inequality doesn't hold is less than $1/2^{L/4}$.*

Proof: Since the keys in $\mathcal{K}_i$ and $\mathcal{K}_j$ are selected according to the bits in random L-bit strings, the expected size of $|\mathcal{K}_i \cap \mathcal{K}_j|$ is $L/2$. By a Chernoff Bound (e.g., see [19]),

$$\Pr(|\mathcal{K}_i \cap \mathcal{K}_j| < L/8) < \left(\frac{4}{e^3}\right)^{L/8},$$

where e is the base of the natural logarithm. Using the approximation $e = 2.71828\ldots$, we can simplify this as

$$\Pr(|\mathcal{K}_i \cap \mathcal{K}_j| < L/8) < \frac{1}{2^{L/4}}.$$

□

So, for example, if we like to guarantee that any two routers share at least 2 keys with 97% likelihood, then we should choose $L \geq 20$. Likewise, if want to guarantee that two routers share $\log n$ keys with probability at least

$$1 - \frac{1}{n^2},$$

then we should choose $L \geq 8 \log n$. Thus, we can use this theorem and the security parameter L to derive bounds on the d-overlap of our key distribution. The next theorem allows us to derive similar bounds on the g-uncensorability of our key distribution.

Theorem 2: *For any subset $\mathcal{K}_i'$ of N keys taken from a set $\mathcal{K}_i$, the expected number of other $\mathcal{K}_j$ sets needed to cover $\mathcal{K}_i'$ is $\Omega(\log N)$.*

Proof: Since the odds of matching a particular key after m tries is $1 - 1/2^m$, the probability of matching all N keys after m tries is

$$\left(1 - \frac{1}{2^m}\right)^N.$$

For this probability to reach $1/2$, m needs to be $\Omega(\log N)$. □

So, we can conclude, then, that a randomly chosen key distribution, as described above, will be likely to be (d, g)-diverse, where we can, for example, choose the parameters so that L is $\Theta(\log n)$, d is $\Omega(\log n)$, and g is $\Omega(\log L)$.

3.5 Dealing with Network Updates

Unlike our leap-frog scheme, the diverse key distribution scheme is quite tolerant of network updates. Adding a new node x to the network requires only that we provide x with L keys so as to maintain the (d, g)-diverse property for the set of distributed keys. The randomized construction described above does this, with high probability (assuming the current number of nodes is proportional to the original number), and it does not require any changes to existing keys. Likewise, adding or removing edges in the network requires no changes to the key sets.

3.6 Evaluation and Analysis

Let us analyze the efficiency and security of our diverse key distribution scheme.

Efficiency. In terms of efficiency, the prime competitor with our key distribution scheme is the key distribution scheme of Eschenauer and Gligor [8]. Their scheme differs from ours in two ways. First, they create a large key pool from which they will sample keys for each device. For example, they advocate creating a key pool of size roughly $10n$ for use with n devices. Our key pool is much smaller, as we advocate a key pool of size $O(\log n)$ for creating a $(\log n, \log \log n)$-diverse distribution of keys. Thus, the key-pool overhead in our scheme improves on that of Eschenauer and Gligor [8] by an exponential factor. To be fair, we should mention that the goals of Eschenauer and Gligor's scheme are different than ours, since their scheme is focused on secure point-to-point message integrity, whereas we are interested in this paper on network broadcast integrity.

Second, Eschenauer and Gligor [8] perform key distribution by having devices select a set of keys randomly from the pool, whereas we assign keys according to a random bit ID assigned to the device. This difference is admittedly subtle, but it allows for the possibility in our scheme that a system manager could use a deterministically-chosen set of error-correcting codes to determine the keys per device, thereby avoiding the use of randomization. We leave as an open problem, therefore, the construction of a set of error-correcting codes that determine a (d, g)-diverse set of keys for large values of d and g.

In terms of implementation, the setup for a broadcast in our scheme using a (d, g)-diverse key distribution requires g hashes, and the verification step requires d hashes, each on strings of size roughly 50 bytes. Thus, we can reuse the estimates from Table 3 to estimate both the setup and verification times for this scheme. For example, if d is 20 and g is 50, then the additional setup time is 36.8 microseconds for SHA1 hashing and the additional time for the verification step is 14.7 microseconds.

Security. As mentioned above, if the nodes in our network have a (d, g)-diverse set of keys, then, in order to inject a spoofed message or modify an existing message, an adversary would have to capture g key sets (or have g nodes collude to perform the requested action). Moreover, in order to falsify the broadcast of a message sent from node i and received by node j, the set of g malicious nodes would have to be positioned along the path from i to j.

4 Applications

In this section we detail how the above data integrity techniques for routing packets can be used in conjunction with simple data validation protocols for securing the well-known link-state and distance-vector algorithms for building routing tables.

4.1 Achieving Integrity in the Setup for Link-State Routing

Having discussed how to efficiently secure a broadcast flooding message, we observe that this approach can be used for the setup of the link-state algorithm. This algorithm is the basis of the well-known and highly-used OSPF routing protocol. In this algorithm, we build at each router in a network G a table, which indicates the distance to every other router in G, together with an indication of which link to follow out of x to traverse the shortest path to another router. That is, we store D_x and C_x at a router x so that $D_x[y]$ is the distance to router y from x and $C_x[y]$ is the link to follow from x to traverse a shortest path from x to y.

These tables are built by a simple setup process, which we can now make secure using the leap-frog or diverse key distribution schemes described above. The setup begins by having each router x poll each of its neighbors, y, to determine the state of the link from x to y. This determination assigns a distance weight to the link from x to y, which can be 0 or 1 if we are interested in simply if the link is up or down, or it can be a numerical score of the current bandwidth or latency of this link. In any case, after each router x has determined the states of all its adjacent links, it floods the network with a message that contains a vector of all the distances it determined to its neighbors. Under our protected scheme, we now perform this flooding algorithm using the leap-frog, chromatic leap-frog, or diverse key distribution methods. Once this computation completes correctly, we compute the vectors D_x and C_x for each router x by a simple local application of the well-known Dijkstra's shortest path algorithm (e.g., see [5, 10]).

Thus, simply by utilizing a secure flooding algorithm we can secure the setup for the link-state routing algorithm. Securing the setup for another well-known routing algorithm takes a little more effort than this, however, as we explore in the next section.

4.2 Achieving Integrity in the Setup for Distance-Vector Routing

Another important routing setup algorithm is the distance-vector algorithm, which is the basis of the well-known RIP protocol. As with the link-state algorithm, the setup for distance-vector algorithm creates for each router x in G a vector, D_x, of distances from x to all other routers, and a vector C_x, which indicates which link to follow from x to traverse a shortest path to a given router. Rather than compute these tables all at once, however, the distance vector algorithm produces them in a series of rounds.

4.2.1 Reviewing the Distance-Vector Algorithm

Initially, each router sets $D_x[y]$ equal to the weight, $w(x, y)$, of the link from x to y, if there is such a link. If there is no such link, then x sets $D_x[y] = +\infty$. In each round each router x sends its distance vector to each of its neighbors. Then each router x updates its tables by performing the following computation:

for each router y adjacent to x **do**
  for each other router w **do**
    if $D_x[w] > w(x, y) + D_y[w]$ **then**
      {It is faster to first go to y on the way to w.}
      Set $D_x[w] = w(x, y) + D_y[w]$
      Set $C_x[w] = y$
    end if
  end for
end for

If we examine closely the computation that is performed at a router x, it can be modeled as that of computing the minimum of a collection of values that are sent to x from adjacent routers (that is, the $w(x, y) + D_y[w]$ values), plus some comparisons, arithmetic, and assignments. Thus, to secure the distance-vector algorithm, the essential computation is that of verifying that the router x has correctly computed this minimum value. We shall use the leap-frog idea to achieve this goal.

4.2.2 Securing the Setup for the Distance-Vector Algorithm

Since the main algorithmic portion in testing the correctness of a round of the distance-vector algorithm involves validating the computation of a minimum of a collection of values, let us focus more specifically on this problem. Suppose, then, that we have a node x that is adjacent to a collection of nodes y_0, y_1, ..., y_{d-1}, and each node y_i sends to x a value a_i. The task x is to perform is to compute

$$m = \min_{i=0,1,\ldots,d-1} \{a_i\},$$

in a way that all the y_i's are assured that the computation was done correctly. As in the previous sections, we will assume that at most one router will be corrupted during the computation (but we have to prevent and/or detect any fallout from this corruption). In this case, the router that we consider as possibly corrupted is x itself. The neighbors of x must be able therefore to verify every computation that x is to perform. To aid in this verification, we assume a preprocessing step has shared a key $k(x)$ with all d of the neighbors of x, that is, the members of $N(x)$, but is not known by x.

The algorithm that x will use to compute m is the trivial minimum-finding algorithm, where x iteratively computes

all the prefix minimum values

$$m_j = \min_{i=0,\ldots,j} \{a_i\},$$

for $j = 0, \ldots, d-1$. Thus, the output from this algorithm is simply $m = m_{d-1}$. The secure version of this algorithm proceeds in four communication rounds:

1. Each router y_i sends its value a_i to x, as $A_i = (a_i, h(a_i||k(x))$, for $i = 0, 1, \ldots, d-1$.
2. The router x computes the m_i values and sends the message $(m_{i-1}, m_i, A_{i-1 \bmod d}, A_{i+1 \bmod d})$ to each y_i. The validity of $A_{i-1 \bmod d}$ and $A_{i+1 \bmod d})$ is checked by each such y_i using the secret key $k(x)$. Likewise, each y_i checks that $m_i = \min\{m_{i-1}, a_i\}$.
3. If the check succeeds, each router y_i sends its verification of this computation to x as $B_i = (\text{“}yes\text{”}, i, m_i, h(\text{“}yes\text{”}||m_i||i||k(x)))$. (For added security y_i can seed this otherwise short message with a random number.)
4. The router x sends the message $(B_{i-1 \bmod d}, B_{i+1 \bmod d})$ to each y_i. Each such y_i checks the validity of these messages and that they all indicated “yes” as their answer to the check on x's computation. This completes the computation.

In essence, the above algorithm is checking each step of x's iterative computation of the m_i's. But rather than do this checking sequentially, which would take $O(d)$ rounds, we do this check in parallel, in $O(1)$ rounds.

5 Conclusion

We have described two techniques—leap-frog packet linking and diverse key distributions—for improving the integrity of network broadcasts and multicasts, and we have given applications of these techniques to the setup algorithms for the link-state and distance vector routing algorithms.

During routing phases, these two techniques offer useful tradeoffs. Leap-frog packet linking adds only two additional values to the payload of a packet and can tolerate no adjacent colluding malicious routers on a path. The diverse key distribution technique, on the other hand, adds up to $O(\log n)$ values to the data payload, but can tolerate small colluding sets of malicious routers. Both of these techniques, however, provide data integrity at low storage and computational overhead per device.

Acknowledgments

We would like to thank Amitabha Bagchi, Luke Bao, and Amitabh Chaudhary, for helpful discussions related to the topics of this paper prior to its acceptance in IEEE S&P, and Radia Perlman, for helpful discussions post acceptance. This research was supported in part by NSF grants CCR-0225642, CCR-0311720, and CCR-0312760.

References

[1] R. Blom. An optimal class of symmetric key generation systems. In *Advances in Cryptography: EUROCRYPT*, volume 209 of *Lecture Notes in Computer Science*, pages 335–338, 1985.

[2] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. In *IEEE Symposium on Security and Privacy*, pages 115–124, 1998.

[3] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *Proc. of the IEEE Security and Privacy Symposim*, pages 197–213, 2003.

[4] S. Cheung. An efficient message authentication scheme for link state routing. In *13th Annual Computer Security Applications Conference*, pages 90–98, 1997.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[6] W. Dai. Speed benchmarks for various ciphers and hash functions. http://www.eskimo.com/∼weidai/benchmarks.html.

[7] W. Du, J. Deng, Y. S. Han, and P. K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *10th ACM Conf. on Computer and Communication Security (CCS)*, pages 42–51, 2003.

[8] L. Eschenauer and V. Gligor. A key management scheme for distributed sensor networks. In *9th ACM Conf. on Computer and Communication Security (CCS)*, pages 41–47, 2002.

[9] J. A. Garay, J. Staddon, and A. Wool. Long-lived broadcast encryption. In *Advances in Cryptology – CRYPTO 2000*, LNCS, pages 333–352. Springer-Verlag, 2000.

[10] M. T. Goodrich and R. Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples*. John Wiley & Sons, New York, NY, 2002.

[11] M. Guerrero-Zapata and N. Asokan. Securing ad hoc routing protocols. In *ACM Workshop on Wireless Security*, pages 1–10, 2002.

[12] R. C. Hauser, T. Przygienda, and G. Tsudik. Lowering security overhead in link state routing. *Computer Networks*, 31(8):885–894, 1999.

[13] Y.-C. Hu, A. Perrig, and D. B. Johnson. Efficient security mechanisms for routing protocols. In *Network and Distributed System Security Symposium (NDSS)*, pages 57–73, 2003.

[14] J. Hwang and Y. Kim. Revisiting random key predistribution schemes for wireless sensor networks. In *2nd ACM Workshop on Security in Ad Hoc and Sensor Networks*, pages 43–52, 2004.

[15] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[16] S. Kent, C. Lynn, J. Mikkelson, and K. Seo. Secure boarder gateway protocol (S-BGP) – real world performance and deployment issues. In *Symposium on Network and Distributed Systems Security (NDSS '00)*, pages 103–116, 2000.

[17] J. Konh, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing robust and ubiquitous security support for mobil ad-hoc networks. In *9th IEEE Int. Conf. on Network Protocols (ICNP)*, pages 251–260, 2001.

[18] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *10th ACM Conf. on Computer and Communication Security (CCS)*, pages 52–61, 2003.

[19] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, 1995.

[20] S. Murphy, M. Badger, and B. Wellington. RFC 2154: OSPF with digital signatures, June 1997. Status: EXPERIMENTAL.

[21] S. Murphy, O. Gudmundsson, R. Mundy, and B. Wellington. Retrofitting security into internet infrastructure protocols. In *DARPA Information Survivability Conference and Exposition (DISCEX)*, pages 3–17. IEEE Computer Society Press, 2000.

[22] S. L. Murphy and M. R. Badger. Digital signature protection of OSPF routing protocol. In *1996 Internet Society Symp. on Network and Distributed System Security (NDSS)*, pages 93–102, 1996.

[23] P. Papadimitratos and Z. J. Haas. Secure link state routing for mobile ad hoc networks. In *Symp. on Applications and the Internet Workshops*, page 379, 2003.

[24] R. Perlman. *Network Layer Protocol with Byzantine Agreement*. PhD thesis, The MIT Press, Oct. 1988. LCS TR-429.

[25] R. Perlman. *Interconnections, Second Edition: Bridges, Routers, Switches, and Internetworking Protocols*. Addison-Wesley, Reading, MA, USA, 2000.

[26] R. L. Rivest. Mit 6.857 computer and network security class notes. http://theory.lcs.mit.edu/∼rivest/notes/notes.pdf.

[27] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer. A secure routing protocol for ad hoc networks. In *10th IEEE Int. Conf. on Network Protocols (ICNP)*, pages 78–89, 2002.

[28] B. Schneier. *Applied Cryptography: protocols, algorithms, and source code in C*. John Wiley and Sons, Inc., New York, 1994.

[29] B. R. Smith, S. Murthy, and J. Garcia-Luna-Aceves. Securing distance-vector routing protocols. In *Symposium on Network and Distributed Systems Security (NDSS '97)*, 1997.

[30] D. R. Stinson, T. van Trung, and R. Wei. Secure frameproof codes, key distribution patterns, group testing algorithms and related structures. *Journal of Statistical Planning and Inference*, 86:595–617, 2000.

[31] B. Vetter, F.-Y. Wang, and S. F. Wu. An experimental study of insider attacks for the OSPF routing protocol. In *5th IEEE International Conference on Network Protocols*, 1997.

[32] S. F. Wu, F.-Y. Wang, Y. F. Jou, and F. Gong. Intrusion detection for link-state routing protocols. In *IEEE Symposium on Security and Privacy*, 1997.

[33] K. Zhang. Efficient protocols for signing routing messages. In *Symposium on Network and Distributed Systems Security (NDSS '98)*, San Diego, California, 1998. Internet Society.

[34] S. Zhu, S. Xu, S. Setia, and S. Jajodia. Establishing pairwise keys for secure communication in ad hoc networks: A probabilistic approach. In *11th IEEE International Conference on Network Protocols (ICNP)*, pages 326–335, 2003.

[35] S. Zhu, S. Xu, S. Setia, and S. Jajodia. LHAP: A lightweight hop-by-hop authentication protocol of ad-hoc networks. In *23rd International Conference on Distributed Computing Systems*, page 749, 2003.