Efficient and Secure Network Routing Algorithms

Michael T. Goodrich

Center for Algorithm Engineering
Dept. of Computer Science
Johns Hopkins University
Baltimore, MD 21218

goodrich@jhu.edu

Abstract

We present several algorithms for network routing that are resilient to various attacks on the routers themselves. Our methods for securing routing algorithms are based on a novel "leap-frog" cryptographic signing protocol, which is more efficient than using traditional public-key signature schemes.

1 Introduction

Routing messages in a network is an essential component of Internet communication, as each packet in the Internet must be passed quickly through each network (or autonomous system) that it must traverse to go from its source to its destination. It should come as no surprise, then, that most methods currently deployed in the Internet for routing in a network are designed to forward packets along shortest paths. Indeed, current interior routing protocols, such as OSPF, RIP, and IEGP, are based on this premise, as are many exterior routing protocols, such as BGP and EGP (e.g., see [5, 9]).

The algorithms that form the basis of these protocols are not secure, however, and have even been compromised by routers that did not follow the respective protocols correctly. Fortunately, all 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 router attacks, for they demonstrate that compromising a single router can undermine the performance of an entire network.

1.1 Security Goals

We are therefore interested in methods for securing routing algorithms against attacks, independent of whether those attacks are malicious or not. Our desire is to design methods that achieve the following properties:

- **Fault detection**. The algorithm should run correctly and, in addition, should detect any computational steps that would compromise the correctness of the algorithm.
- **Damage containment**. The algorithm should contain the damage caused by an incorrect router to as small an area of the network as possible.
- **Authentication**. The algorithm should confirm that each message is sent from the host or router that the message identifies as its source.
- **Data integrity**. The algorithm should confirm that the contents of received messages are the same as when they are sent, and that all components of a message are as intended by the algorithm (even those message portions added by routers other than the original sender).

- **Timeliness**. The algorithm should confirm that all messages interacting to perform this algorithm are current up-to-date messages, thereby preventing replay attacks.

We are not explicitly requiring that we also achieve confidentiality, since this can easily be achieved by encrypting the sensitive content of a message. For example, message content encryption can be achieved in the application layer or by using services in the IPSec protocol (which does not address routing security, just end-to-end message authentication and confidentiality).

1.2 Prior Related Work

Routing security was first studied in the seminal work of Perlman [8] (see also [9]), who studied flooding and shortest-path routing algorithms that are resilient to faulty routers. Here 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, for example, to design a secure version of the flooding algorithm, which can be further used to design a secure algorithm for the setup phase of link-state routing. Moreover, as we will show, a signature-based approach can also be used to design a secure distance-vector routing setup algorithm as well. Even so, several researchers have commented that, from a practical point of view, requiring full public-key signatures on all messages is probably not efficient. Signing and checking signatures are expensive operations when compared to the simple table lookups and computations performed in the well-known routing algorithms. Nevertheless, Murphy *et al.* [7, 6] discuss some of the details of a protocol that would implement such a scheme. Likewise, Smith *et al.* [11] discuss how to extend a signature-based approach to distance-vector algorithms.

Motivated by the desire to create efficient and secure routing algorithms, several researchers have recently designed routing algorithms that achieve routing security at computational costs that are argued to be superior to those of Perlman. Given that the signature-based of Perlman is already highly-secure, this recent research has used fast cryptographic tools, such as hashing, instead of 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, for this new line of research in routing security is to create practical and secure routing algorithms by introducing natural assumptions on the network and its attackers while also using fast cryptographic tools to secure the routing algorithms under these assumptions.

Cheung [2] shows how to use hash chaining to secure routing 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 *et al.* [4] 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 require that the routers in the network be synchronized. Zhang [14] 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.

As will be the focus in this paper, all of these previous papers focused on the issue of how to robustly perform flooding protocols and set up the routing tables for link-state algorithms. Also of related interest, is work of Bradley *et al.* [1], who discuss ways of improving the security of packet delivery after the routing tables have been built. In addition, Wu *et al.* [13] and Vetter *et al.* [12]

discuss some practical and empirical issues in securing routing algorithms. Of specific interest in their work is their observation that a single bad router can adversely affect an entire network for as much as an hour or more.

1.3 Our Results

In this paper we describe a new approach to securing the setup and flooding stages of routing algorithms. After a preliminary setup that involves distributing a set of secret keys equal that total no more than the number of routers, our method uses simple cryptographic hashing of messages (HMACs) to achieve security. Our approach involves the use of a technique we call "leap-frog" message authentication, as it allows parties in a long chain to authenticate messages between every other member in the chain. Using this approach, we show how to secure flooding, link-state, and distance-vector algorithms, under the reasonable assumption that no two bad routers are colluding and are within two hops of each other. Such a strategy would even be effective for routing in Gnutella networks, which are notoriously insecure but experience few, if any, insider collusion attacks. Our algorithms can also be used in multicast routing, for they 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. We describe the main details of our leap-frog approach to router security in the sections that follow.

2 Flooding

We begin by discussing the flooding protocol and a low-cost way of making it more secure. Our method involves the use of a novel "leap frog" message-authenticating scheme using cryptographic hashing.

2.1 The Network Framework and the Flooding Algorithm

Let $G = (V, E)$ be a network whose vertices in V are routers and whose edges in E are direct connections between 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. Furthermore, we assume that G is biconnected, that is, that it would take at least two routers to fail in order to disconnect the network. This assumption is made both for fault tolerance, as single points of failure should be avoided in computer networks, and also for security reasons, for a router at an articulation point can fail to route packets from one side of the network to the other without there being any immediate way of discovering this abuse.

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 table S_x that stores the largest sequence number encountered so far from each possible 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 is 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 router 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., [13, 12]), although many such failures can be considered router misconfiguration 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 Securing the Flooding Algorithm on General Networks

On possible way of avoiding the possible failures that compromised or misconfigured routers can inflict on the 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. Unfortunately, this approach is computationally expensive. It is particularly expensive for overall network performance, for, as we discuss later in this paper, flooding is often an important substep in general network administration and setup tasks.

Our scheme is based on a light-weight strategy, which we call the *leap-frog* strategy. The initial setup for our scheme involves the use of a public-key infrastructure, but the day-to-day operation of our strategy takes advantage of much faster cryptographic methodologies. 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. This key is created in a setup phase and distributed securely using the public-key infrastructure to all the members of $N(x)$. 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 [10]. 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 most one router is compromised 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 each w that is its neighbor 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 as in the insecure version of the flooding algorithm, the first time a router w receives this message, it can process it, updating the sequence number for s and so on.

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 considered 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. Likewise, should a router choose to not send the message, then the message will still arrive, by an alternate route, since the graph G is biconnected. The message will be correctly processed in this case as well, since a router is not expecting messages from s to arrive from any particular direction. That is, a router x does not have to wait for any other messages or verifications before sending in turn a message M on to x's neighbors.

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 degree(x) hash computations, where degree(x) denotes the degree of x. Typically, for communication networks, the degree of a router is kept bounded by a constant. Thus, this work compares quite favorably in practice to the computations that 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 Trading Message Size for Hashing Computations

In some contexts it might be too expensive for a router to perform as many hash computations as it has neighbors. Thus, we might wonder whether it is possible to reduce the number of hashes that an intermediate router needs to do to one. 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 to propagate the flooding message. 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 case, we change the preprocessing step to that of computing a small-sized coloring of the vertices in G so that no two nodes are assigned the same color. Algorithms for computing or approximating such colorings are known for a wide variety of graphs. For example, a tree can be colored with two colors. Such colorings might prove useful in applying our scheme to multicasting algorithms, since most multicasting communications actually take place in a tree. A planar graph can be colored with four colors, albeit with some difficulty, and coloring a planar graph with five colors is easy. 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 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 $h_i = h(s|j+1|M|k_i)$. (As a side note, we observe that the prefix of the bit string being hashed repeatedly by s is the same for all hashes, and its hash value in an iterative hashing function need only be computed once.) 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 $(s, j+1, M, h_1, h_2, \ldots, h_c)$ 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. In this case, the message is authenticated, since y could not manufacture the value of h_i.

If the graph G is biconnected, then even if one router fails to send a message to its neighbors, the flood will still be completed. Even without biconnectivity, 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.

3 Setup for Link-State Routing

Having discussed how to efficiently secure the flooding algorithm, let us next turn to a point-to-point unicast routing algorithm—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 scheme 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 or chromatic leap-frog method. 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 [3]).

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 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.

4.1 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 again the leap-frog idea to achieve this goal.

4.2 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.

References

[1] 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.

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

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[4] R. Hauser, T. Przygienda, and G. Tsudik. Reducing the cost of security in link-state routing. *Computer Networks and ISDN Systems*, 1999.

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

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

[7] S. L. Murphy and M. R. Badger. Digital signature protection of OSPF routing protocol. In *Proceedings of the 1996 Internet Society Symposium on Network and Distributed System Security*, pages 93–102, 1996.

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

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

[10] B. Schneier. *Applied cryptography: protocols, algorithms, and sourcecode in C*. John Wiley and Sons, Inc., New York, 1994.

[11] 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.

[12] 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.

[13] 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.

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