2005-04-13

Improving Routing Security Using a Decentralized Public Key Distribution Algorithm

Jeremy C. Goold
Brigham Young University - Provo

IMPROVING ROUTING SECURITY USING A DECENTRALIZED PUBLIC KEY DISTRIBUTION ALGORITHM

by

Jeremy C. Goold

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

April 2005

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Jeremy C. Goold

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Mark Clement, Chair
Date	Kent E. Seamons
Date	Kevin Seppi

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Jeremy C. Goold in its final form and have found that (1) its format, citations and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Mark Clement
Chair, Graduate Committee

Accepted for the Department

David W. Embley
Graduate Coordinator

Accepted for the College

G. Rex Bryce
Associate Dean,
College of Physical and Mathematical Sciences

ABSTRACT

IMPROVING ROUTING SECURITY USING A DECENTRALIZED PUBLIC KEY DISTRIBUTION ALGORITHM

Jeremy C. Goold

Department of Computer Science

Master of Science

Today's society has developed a reliance on networking infrastructures. Health, financial, and many other institutions deploy mission critical and even life critical applications on local networks and the global Internet. The security of this infrastructure has been called into question over the last decade. In particular, the protocols directing traffic through the network have been found to be vulnerable. One such protocol is the Open Shortest Path First (OSPF) protocol. This thesis proposes a security extension to OSPF containing a decentralized certificate authentication scheme (DecentCA) that eliminates the single point of failure/attack present in current OSPF security extensions. An analysis of the security of the DecentCA is performed. Furthermore, an implementation of DecentCA in the Quagga routing software suite is accomplished.

ACKNOWLEDGMENTS

Thanks to all who aided me in preparing this work. Many helped me along the way, likely without even knowing it. I thank my advisor, Dr. Clement, for his direction, insight, and most importantly his ever present positive encouragement. Spencer Cox was very helpful in listening, and often correcting, my reasoning and thought process. Sincere gratitude also goes to my wife, Taffy, for her unwavering support and love. Finally I could not have even begun without the support of a loving Father in Heaven.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In today's society, the Internet is used to support a number of important services. Financial institutions use the Internet for online banking and trading. Consumers use it for purchasing goods and services. Companies with satellite offices use the Internet as part of their corporate network infrastructure. Clearly this global network infrastructure has grown to be a critical resource in the eyes of many in society today. Existing network layer security has not been widely used to protect these critical resources. This thesis develops a decentralized security mechanism to protect network layer routing in the Open Shortest Path First (OSPF) routing protocol.

While the Internet infrastructure was designed to be extensible and reliable, the Internet and other networks like it were not originally constructed with security in mind [17, 29]. In addition, the Internet's design actually makes it particularly vulnerable to malicious attack [3]. Much research today has focused on enforcing security from an application standpoint. For example, the secure socket layer (SSL) protocol provides end to end authentication and encryption to network based applications. One primary use for SSL is securing transactions on the world wide web. Users now have some confidence in the integrity and confidentiality of data being transported between applications.

Users may not, on the other hand, have confidence in the ability of the network infrastructure itself to securely deliver information to its proper destination [4] in the face of malicious attackers. Modern networks use a number of protocols to direct the flow of traffic from one endpoint to another. These protocols do occasionally support some sort of authentication to create trust between protocol participants. However, these security features are seldom used and are often insufficient to provide adequate protection.

1.1 Open Shortest Path First (OSPF) Protocol

OSPF is a major network routing protocol used in computer networks. In order to analyze its security properties and understand the potential for improvement in OSPF security, the reader must have a basic understanding of the protocol itself. The following section presents these fundamental concepts.

OSPF is an intra-domain routing protocol. This means that it is used within a single autonomous system (AS) to route traffic. Other protocols, such as the border gateway protocol (BGP) are used to route traffic between AS's.

The link-state paradigm is used by OSPF to compute routes through the network. In link-state routing, each router advertises knowledge only about directly attached links. Information about each of these links is disseminated throughout the OSPF routing area using a flooding protocol. When a router receives link-state information from a neighbor, it retransmits this information to all other directly attached neighbors. Through this method, all participating routers have complete knowledge about the structure of the network topology. With this database of information about the topology, known as the link-state database, each router can compute the shortest path to every other participating router.

When the state of a link changes, routers attached to this link share this information

with all other neighbors. Information about link state change is then flooded as before so that the entire network is notified of the change in the topology. OSPF is dynamic. Changes in the network topology are detected and knowledge of the changes are shared throughout the network. This results in shortest paths being recomputed by each participating router [15].

1.2 Current OSPF Security

1.2.1 OSPF Neighbor to Neighbor Security

Security in OSPF version 2 is specified only for exchanges between neighboring routers [15]. This feature is implemented in many widely available routing platforms, including Cisco's IOS [27], Foundry's enterprise class routers [1], as well as the open source Quagga routing platform [2]. Three different approaches to insuring authenticity and integrity of routing messages between neighbors are specified in OSPF v2:

- None: No authentication or integrity assurance is used.
- Simple Password: A clear text password is transmitted between neighbors to insure authenticity of messages.
- Cryptographic Authentication: A keyed hash, or message authentication code (MAC) is transmitted to ensure authenticity and integrity.

The authentication of neighbor exchanges is important to the security of an OSPF network. Authenticating neighbor exchanges ensures that foreign devices cannot suddenly appear on the network without at least one other current network node trusting it. Furthermore, when a message authentication code is used, foreign devices may not modify the content of information passing between neighbors. Clearly, cryptographic

authentication provides the highest level of assurance and is the least susceptible to compromise. Plain text passwords are vulnerable, because an attacker with access to the link between two routers can observe the password. The attacker can then use the compromised password to masquerade as one of the routers.

This neighbor to neighbor security does not provide authentication or integrity assurance for information that is flooded throughout the network as the data must pass though multiple neighbor pairs. If a valid router becomes subverted or begins malfunctioning, it may flood incorrect data about its connected links, or it may modify information sent by other routers, or even impersonate other routers by originating data appearing to come from another router. This results in the poisoning of routing tables for other routers in the domain. A poisoned routing table is one that contains information that does not reflect the actual topology of the network. Chakrabarti [4] proposes that the results of routing table poisoning include:

- **Suboptimal Routing:** Real time or soft real time applications such as video conferencing and IP Telephony require a certain level of network performance in order to operate acceptably. Incorrect routing information may cause data to travel on paths that are either longer or more congested than paths that would be selected by the routing protocol. This may result in degraded network performance and thereby unacceptable operation of these real time applications.

- **Congestion:** Incorrect routing information may cause flows of data that would otherwise have been forwarded on different links to be forwarded on the same link. This could cause data to be lost as routers on either end of the congested link drop packets. As this congestion is artificial, normal congestion control mechanisms will not solve this problem.

- **Partition:** A partition occurs when one portion of a network is unable to communicate with another portion of the network. Incorrect routing information

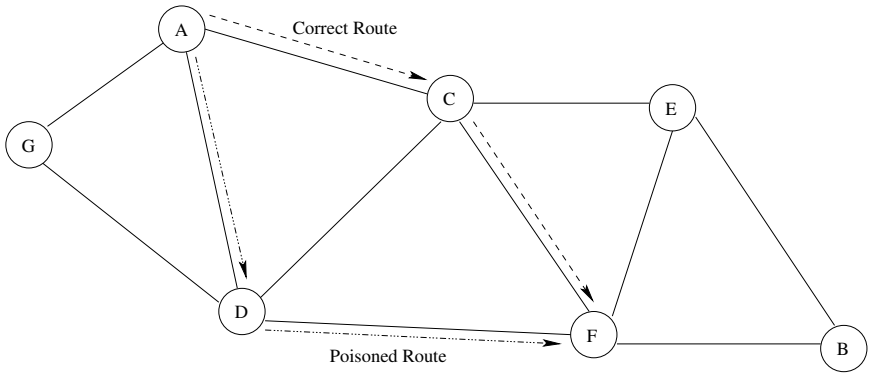


Figure 1.1: Routing table poisoning example.

may cause links that normally connect these two portions of the network to appear as though they are down.

- **Overwhelmed Host:** This type of problem may occur if routing table information sends numerous packets destined for various different hosts to a single host. The resulting traffic may then overwhelm the victim host causing services hosted by the victim to become unavailable.

- **Looping:** Looping occurs when packets are repeatedly forwarded between the same set of routers. This can be caused when router A sends data to router B, which sends data to router C, which sends data back to router A. These packets will continue in this loop until the network considers them too old, such as when the time to live field expires in TCP/IP.

- **Access to Data:** Routing table poisoning may cause traffic to be redirected through a malicious router. This router can then monitor possibly sensitive data. This gives individuals access to data that would otherwise be inaccessible.

An example of router table poisoning is illustrated in Figure 1.1. Assume node A ordinarily communicates with node F through node C. If node D wishes to have access to that data, it could attempt to poison node A's routing table. In an OSPF network, node D could impersonate node C by initiating a link state update packet

that appears to originate from C indicating link (C, F) is unavailable. This would cause node A to reroute traffic destined for node F through node D, thereby giving node D access to this data. This scenario can occur because node A has no way of knowing the real source of the forged link state advertisement.

1.3 Current Proposals for OSPF Security Extensions

In order to overcome this need for end-to-end authentication of link state information, Murphy et al. [16] propose a protocol for signing link state updates flooded throughout the network. A certificate for the signing router, containing its public key and role in the routing infrastructure, is disseminated through the network in the same fashion as link state updates. This certificate is signed by a centralized certificate authority (CA). Each router in the OSPF routing area is configured with the public key of the CA. The CA's public key is used to verify the router certificate. Once the certificate is verified, the router's public key can be used to verify the origin and integrity of link state updates.

Although these extensions could solve many security problems, few implementations use digitally signed OSPF. While certainly not authoritative, it is interesting to note that of the three implementations mentioned above only the Foundry routers claim compliance with the OSPF digital signature specification. There are a number of possible reasons for this. Fundamentally, the relative lack of acceptance involves both implementation difficulties and cultural notions. The major implementation and cultural challenge is that the digital signature extension requires an established and centralized public key infrastructure. A centralized certificate authority is problematic for a few reasons:

- it provides a single point of failure and attack [31],

- it is not feasible for domains that cross organizational boundaries [14] that may not all trust each other, and
- the culture of the Internet is generally opposed to any kind of centralized control.

1.4 Thesis Statement

A decentralized certificate authority infrastructure can be used to secure network layer protocols such as OSPF.

1.5 Proposed System Benefits

This research defines a modification of the digital signature extension proposed by Murphy et al. [16] to provide a decentralized router certificate authentication scheme (DecentCA). This system has the following benefits:

- It eliminates the need for a centralized certificate authority.
- A graph of the trust relationships between routers is used to identify correct public key bindings.
- A victim is able to directly identify its attacker(s).

An analysis of the effects of modification on the security properties of the algorithm is also performed.

Chapter 2

Cryptographic Building Blocks

Building security into a system requires the use of basic cryptographic building blocks. These building blocks include symmetric cryptography, asymmetric cryptography, and hash functions. From these building blocks, data authentication, integrity, and confidentiality is provided.

2.1 Symmetric Cryptography

Traditional cryptography dates back to the time of Caesar and the ancient Romans. At this time, cryptography consisted of obfuscating a message, the plaintext, to obtain ciphertext using a key shared by the sender and receiver. The receiver can then use the shared key to decode the message. While the obfuscation methods have become much more complex, using a well known algorithm with a secret key is still common today.

Symmetric cryptography is performed using a single parameter family of invertible transformations. Most symmetric cryptographic systems such as the Data Encryption Standard (DES), 3-DES [18], and the Advanced Encryption Standard [20] involve transformations including byte substitution, translation, and bitwise operators like XOR. The parameter for the transformation is the shared key. Each cryptographic

algorithm is well known, so the real security in the algorithms is derived from the secrecy of the key.

Using a shared key in large groups, however, is cumbersome. The number of shared keys to be managed becomes overwhelming. For a situation with n individuals each wishing to communicate securely, the number of keys required is $n(n-1)/2$. Each of the n individuals needs a different key to communicate with each of the $n-1$ other individuals. Because the keys are shared between two individuals, only half of the $n(n-1)$ keys are needed.

Managing the shared keys is known as *key management*. Key management is a challenge in all encryption schemes, but with symmetric encryption it becomes extremely difficult. In the scenario discussed above, the $n(n-1)/2$ keys must be communicated secretly and without modification between communicating individuals. Traditionally this might mean some type of *out of band* communication where the channel used to communicate the keys is separate and distinct from the channel being secured with the keys. Out of band communication is generally a more secure but much more difficult means of communication. For secure communications between computers, keys may be exchanged by system administrators over the phone.

2.2 Public Key Cryptography

The difficulty in solving the key management problem led Whitfield Diffie and Martin Hellman to pursue another direction in cryptography known as asymmetric or public key cryptography [7]. In public key cryptography encryption and decryption are performed using two distinct keys known as the public and private keys, respectively. The public key class of algorithms, such as the Diffie Hellman key exchange [25], RSA [12], and the Digital Signature Standard [19] rely on two single parameter transformations, one for encryption and one for decryption. The transformations used in

public key cryptography are fundamentally different than those used in symmetric key cryptography. Transformations include basic arithmetic operations, like multiplication and exponentiation, on a finite field.

The advantage of a public key cryptosystem is that the key used to encrypt (commonly known as the public key) can be published for use by anyone, while the key used to decrypt (commonly known as the private key) must be kept secret. This ability results from relatively large (around 1024 bytes) key sizes. It is currently computationally infeasible to derive the private key from the public key.

The ability to disclose the public key to anyone simplifies the key management problem. Instead of $n(n-1)/2$ keys, and the system only requires n keys be distributed (each individual's public key).

2.3 Hash Functions

Hash functions are a cryptographic building block used most often in ensuring message integrity. A hash function H is used to create a fingerprint of a block of data. According to Stallings, a secure hash function most generally has the following six characteristics [28]:

1. H can be applied to a block of any size.
2. H produces a fixed length output.
3. $H(x)$ is relatively easy to compute for any given x, making hardware and software implementations practical.
4. For any given value h, it is computationally infeasible to find x such that $H(x) = h$. This is sometimes referred to in the literature as the one-way property.
5. For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$. This is sometimes referred to as weak collision resistance.

6. It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$. This is sometimes referred to as strong collision resistance.

Given these features, hash functions are used in a number of different security applications.

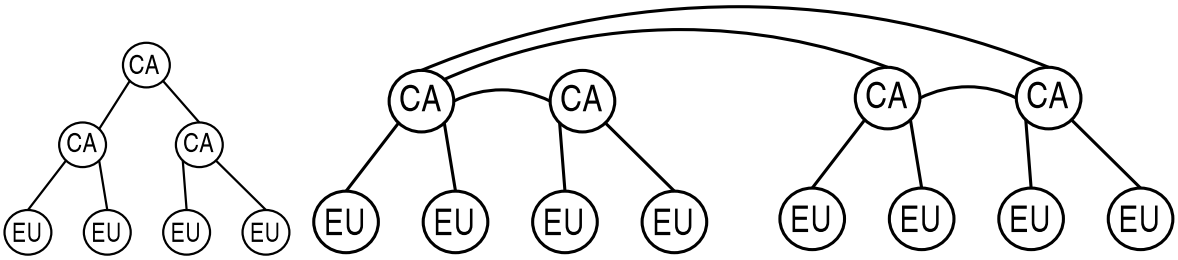
2.4 Digital Signatures

One such application is digital signatures. A digital signature is used to authenticate a message and verify that it has not been modified since the time the signature was created. The signature is generated by the sender by computing the hash of the message and encrypting the hash with the sender's private key. The receiver of the message verifies the signature by decrypting the hashed value sent with the message, computing again the hash of the message, and comparing the computed hash value with the decrypted hash value. If the two match, the message is from the purported sender and has not been modified.

The receiver can be confident of these two facts because an attacker would have accomplish one of two tasks: generate a message that hashes to the same value as the original message, or encrypt a new hash corresponding to the modified message. The first task is improbable due to property 4 of a secure hash function, and the second task is improbable if the attacker does not possess the sender's public key. In this way, digital signatures provide an effective way of verifying the authenticity and integrity of messages if individual public keys can be distributed in a reliable fashion.

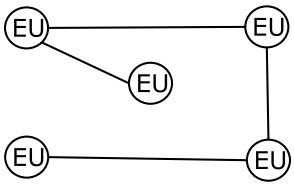
2.5 Public Key Infrastructure

A public key infrastructure (PKI) is used to reliably distribute public keys. The receiver of a public key must be able to verify the identity of the owner of the public



(a) **Hierarchical PKI**: End users (EU) receive public key certificates issued by (possibly delegated) CA's arranged in a tree structure.

(b) **Oligarchical PKI**: End users (EU) receive public key certificates issued by (possibly delegated) CA's arranged in a tree structure with multiple possible roots.



(c) **Anarchy PKI**: End users (EU) receive public key certificates issued by other end users.

Figure 2.1: Three different PKI trust models.

key (i.e., the identity of the individual possessing the private key associated with the received public key). PKI's consist of *certification authorities* that sign *certificates* to bind an individual's name to a public key. There are a few different approaches to organizing a PKI [22], as illustrated in Figure 2.1.

The first approach is a hierarchy. At the root of the hierarchy is a single certificate authority (CA). This root CA may be responsible for signing all certificates, or the root CA may delegate the certificate signing authority to other CA's. A chain is built up from the receiver of the certificate (the individual owning the public key) to the root CA through delegated CA's. A second approach is an oligarchy of CA's. In an oligarchy, multiple root CA's trust each other. This spreads the trusted role of CA's

and may be useful in scenarios where no single individual or organization is trusted by everyone. A third approach to organizing a PKI is an anarchy. Any individual in an anarchy PKI can take on the role of a CA. This organization is most commonly associated with the Pretty Good Privacy (PGP) system used to secure email [32].

Chapter 3

Related Work

Much work relating to routing protocol security has been performed. This chapter covers the portion of work most closely related to this thesis. Efforts to define attack models against network routing occurred first. To counter these attacks, secure routing protocols were then developed, but were deemed practically infeasible due to the computational overhead involved. Research followed to decrease this overhead. In addition, some work to decentralize routing security has been performed.

3.1 Attack Models

When discussing any type of security it is important to analyze the attack models to which the system is vulnerable. Chakrabarti and Manimaran [4] discuss various attack models pertaining to the Internet. There are two classes pertaining to link state routing: proactive attacks, and inactive attacks. Proactive attacks consist of actions taken by a malicious router, including changing link cost, link addition, and link deletion. Inactive attacks involve the absence of actions that take place under normal circumstances, including: ignoring a change in link cost, ignoring link deletion, and ignoring link addition. In most cases, the goal is to cause traffic to flow in a way that reveals information or causes congestion.

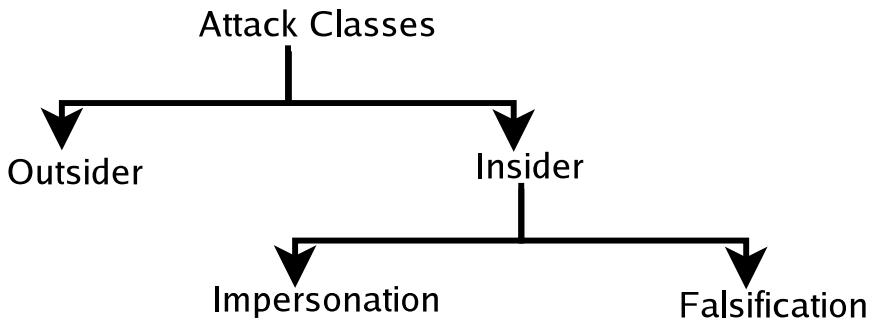


Figure 3.1: Attack class tree.

Altering traffic flow is a fundamental characteristic of network layer attacks. They do not, in general, modify application data or gain access to data directly. Instead these attacks attempt to disrupt the application's ability to communicate. In this way network layer attacks are one step removed from direct application attacks. Network layer attacks are therefore much more insidious and difficult overcome. This is particularly true because the network layer was not originally designed with security in mind. Additionally, the distributed nature of the Internet, and computer networks in general, make the problem of devising and implementing security measures much more difficult.

Other research divides possible routing attacks into two categories (Figure 3.1), outsider attacks and insider attacks [11, 29, ?]. Outsider attacks are perpetuated by devices that are not authorized to participate in the routing protocol. For instance, in the OSPF protocol, OSPF packets are regular IP packets of type 89. These packets can be generated anywhere on the Internet with a destination address of a valid OSPF router. If this IP address is a public address, and system administrators have not configured firewalls to block these packets, a properly formatted OSPF packet can travel across the Internet unimpeded to the destination router. This router then accepts this as a valid routing packet. Insider attacks, on the other hand, are conducted by devices that are authorized to participate in the routing protocol. An insider, on the other hand, is an authorized, but subverted router.

Huang et al. [11] further divide insider attacks into two classes: a router originating incorrect information for itself (falsification), and a router originating incorrect information for other routers (impersonation). The first class is very difficult to overcome because the very nature of routing protocols dictates that each router advertise information to which it alone may be privy. If a router begins to lie about its surroundings, the only solution is to require an outside authority to verify the correctness of advertised information. The second class of insider attacks is still difficult, but can be overcome in a much more straightforward fashion because once information is flooded by the originating router, many more devices become privy to the correct data.

3.2 Digitally Signed Routing Updates

The initial work in overcoming the attack models discussed above was performed by Radia Perlman in her Doctoral thesis titled Network Layer Protocols With Byzantine Robustness [21]. This thesis first defines the levels of robustness for a network layer protocol in the face of simple failures, where nodes cease operation, and Byzantine faults, where nodes may arbitrarily malfunction:

- **Simple robustness**: The network layer will function properly when routers are simply removed from the network. This may occur when a device ceases operation due to a power outage or hardware failure.
- **Self stabilization**: Following a Byzantine fault or simple failure, the network is guaranteed to operate correctly if the faulty node is removed.
- **Byzantine detection**: Byzantine faulty nodes are correctly identified by the network.
- **Byzantine robustness**: The network operates correctly in the face of Byzantine faults.

These properties are orthogonal in nature. That is to say, a network may exhibit Byzantine robustness but not Byzantine detection.

Perlman describes how Byzantine robustness can be achieved using a robust flooding protocol. Flooding is the simplest form of routing; each message is forwarded to all neighbors, except the neighbor from which the message was received. As a result, the message traverses every link in the network, and every path from the originating router to other routers is traversed. Assuming that some type of fair queuing is used at each router, the message is guaranteed to be delivered to every node in the network. Furthermore, to insure integrity of messages, Perlman proposes attaching a digital signature to each message. The public keys used to verify the signature of a message are distributed using a "trusted node" service. The "trusted node" disseminates a list of public keys throughout the network. Each router in the flooding process then verifies the packet's signature before forwarding it. Perlman uses this robust flooding protocol to distribute control messages in a robust link state routing protocol.

Murphy et al. [16] base their OSPF with digital signatures specification discussed in Section 1.3 on Perlman's work.

3.3 Improving Routing Security Performance

While the performance of routing security enhancements is not the central focus of this thesis, the feasibility of using digital signatures in routing has presently been severely hampered by the overhead resulting from digital signature generation and verification.

The digital signature approach is computationally expensive. Computing signatures and verifying signatures incurs significant overhead. Asymmetric cryptography is on the order of 1000 times slower [25] than symmetric cryptography. Important work related to this thesis includes efforts to improve performance of different routing

security algorithms. Much of the work has involved replacing public key signing and verification with hashing functions where possible in the algorithms.

3.3.1 Lamport's Hash

Hauser et al. [10] discuss how Lamport's hashing algorithm [13] can be used to reduce the frequency of signing and verifying link state update messages. Lamport's hashing algorithm for authentication can be summarized as follows:
If Alice wishes to authenticate herself to Bob, and Alice and Bob have agreed upon a secure hash function, the following transactions occur:

1. Alice selects a secret seed value R and a number n.

2. Alice computes $H^n(R)$. That is to say, Alice computes $H^0(R) = R$, $H^1(R) = H(H^0(R))$, $H^2(R) = H(H^1(R)))$, $\ldots$, $H^n(R) = H(H^{n-1}(R))$.

3. Alice initially communicates $H^n(R)$ and n to Bob in such a way that Bob can be assured it came from Alice and has not been modified.

4. When Alice wishes to authenticate herself to Bob, she sends her identification to Bob.

5. Bob responds with a challenge n.

6. Alice responds with $H^{n-1}(R)$.

7. Bob computes $H(H^{n-1}(R))$ and compares it with his stored value $H^n(R)$. If the values match, Alice is authenticated.

8. Bob replaces his stored value of $H^n(R)$ and n with $H^{n-1}(R)$ and $n-1$, respectively.

Because the hash function is one way and R is kept secret by Alice, Bob can be assured that only Alice is capable of computing $H^{n-1}(R)$. Also, because the hash function is collision resistant, no other value X would result in $H(X) = H^n(R)$.

Hauser proposes to use this capability in sending routing updates. When an initial link state update is distributed by a router, an anchor value, $H^n(R), n$ is also sent. The integrity and authenticity of these values are guaranteed using a standard digital signature. Future link state updates can be of two forms, depending on the information contained in the update. If the update is communicating new information (the state of the link has changed), a new digitally signed link state update is sent including a new anchor value. However, if the state of the link has not changed and the update is simply a periodic refresher update, the value $H^{n-1}(R)$ is sent in place of the full link state update.

Routers receiving this update can compute $H(H^{n-1})$ and compare it with the stored anchor value. If the values match, the message is authentic and is taken to indicate that the state of the particular router's links have not changed. The cost of computing this hash is significantly lower than the cost of verifying a digital signature.

3.3.2 Limited Key Distribution

Goodrich proposes a different use for hashing to improve routing security performance [9]. The basis of this approach involves *leapfrog authentication*. For every path through the network, every other next-hop router shares a symmetric key. A hash of the message and the shared key is generated. Two such hashes are appended to the message. In this way each router in the message's path can assure that the message has not been changed by the previous router.

In Goodrich's proposal for secure routing, the neighbors of router s each share a symmetric key, k(s), which is unknown to s. The symmetric key is initially distributed using the public keys of the routers involved. This symmetric key exchange between

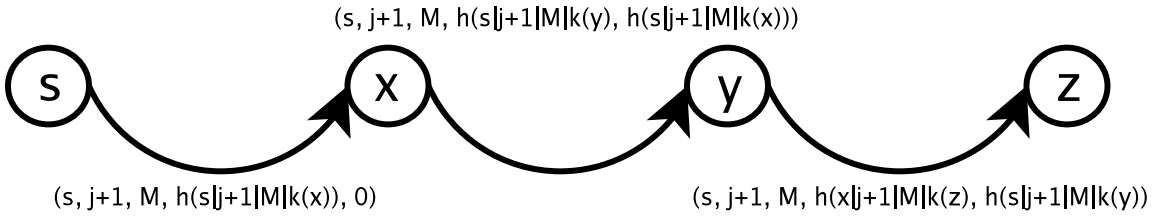


Figure 3.2: Limited key distribution example. Router s floods a message to neighbor x including a hash of the message and symmetric key shared by x's neighbors. Router x can verify the message directly, but once the message is passed to router y, y needs to verify that x has not modified the message. Router y does this by generating a hash of the message contents and the symmetric key shared by x's neighbors. Router x could not have modified the message and generated this hash because it has no knowledge of the key shared by all its neighbors. At the same time, x has generated a hash of the message and the secret key shared by all of y's neighbors. This hash can be used by z to verify that y has not modified the message.

immediate neighbors is performed for every router in the network.

When s wishes to flood a message, M, through a neighboring router x, it sends:

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

where s is the identity of the originating router, $j+1$ is the next sequence number to be used by the flooding algorithm, M is the message itself, $h(s|j+1|M|k(x))$ is the hash of the originating router's id, the sequence number the message itself and the key shared by all of x's neighbors.

Router x will receive this message and forward it on to another of its neighbors. Because x received this message directly from s, it can be verified using standard procedures. When y, another neighbor of x, receives this message it can verify that x did not modify the message by generating the value $h(s|j+1|M|k(x))$ and ensuring that it matches the value placed in the message by s. Router x does not have knowledge of $k(x)$, and could not have modified the message and generated a valid hash value.

Furthermore, the message that x forwards to its neighbor y contains the value $h(s|j+1|M|k(y))$ as well as the value $h(s|j+1|M|k(x))$. When y forwards the message on to z, z can verify that y did not modify the message by generating the value $h(s|j+1|M|k(y))$ and ensuring the value matches the value in the message. Again, y could not have generated this message because it does not know $k(y)$.

This method prevents messages from being modified by single routers. However if multiple neighboring routers collude to modify a message, Goodrich's approach can be overcome. In the scenario discussed above, if x and y are colluding, y could simply fail to verify the hashed value, and because z has no way of verifying that x did not modify the message, the modified message continues to flood undetected.

Ming Yu extends this idea by requiring both the hop one (direct neighbors) and hop two (next neighbors) neighbors share a symmetric key [30]. This increases the number of sequential neighbors to two by including three hash values in the message. The same process is followed by both the hop one and hop two neighbors to verify the integrity of messages.

Both of these approaches have the advantage that modified messages do not flood throughout the network but are discarded immediately when the modification is discovered. However, both require the use of a centralized public key infrastructure to initially distribute shared keys.

3.4 Peer to Peer Public Key Infrastructures

Dheberi proposes a public key infrastructure similar to what is presented in this research. It is a distributed peer-to-peer PKI. Like many peer-to-peer systems all nodes in the PKI act as both clients wishing to verify certificates and certificate authority with the ability to verify certificates. Dheberi's system, however, is a general tool intended for the end user and not applied to any particular application. In

addition, and most importantly, the proposal contains no analysis as to the security system. No mention is made regarding what security properties the system contains or what attacks it is meant to defend against. DecentCA is a system similar to Dheberi's. However instead of being a generic end user application, DecentCA is a protocol defined to secure OSPF. This yields particular security problems that must be analyzed and solved by DecentCA.

Chapter 4

Algorithm Development

This chapter describes the decentralized certificate authentication algorithm. DecentCA uses a number of concepts introduced in Pretty Good Privacy (PGP) [32]. PGP can be used to authenticate email messages. Email is digitally signed by the sender using a private key, and the message can be verified by the receiver using the sender's public key. The public key is obtained either from the sender out of band or from a public key server.

DecentCA differs from PGP because it does not use a central key server(s). In PGP, all public keys are stored in a single key server and automatically copied to *mirror* key servers. In DecentCA, keys are dynamically distributed using the OSPF reliable flooding protocol. In this way, the entire network is collectively responsible for acting as the key server. No single server provides a point of failure or attack.

4.1 Trust Graphs

Trust graphs are a fundamental concept in DecentCA and are very similar to the concept used in Pretty Good Privacy (PGP) [32]. A trust graph T is a directed graph containing verticies and edges, (V_T, E_T). Verticies are network nodes, such as routers and their associated public key. An edge from node s to node r is a digital

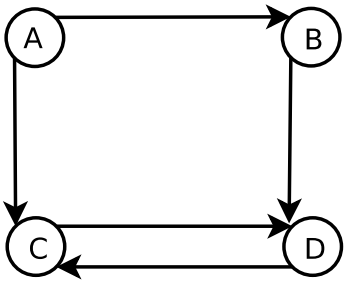


Figure 4.1: An example trust graph.

signature of node r's identity to public key binding by s. This is a statement made by s asserting the value of r's public key.

Trust graphs can then be used to identify the public key of an entity in the graph by locating the node containing the entity's identity. For example, consider the graph shown in Figure 4.1. When A receives a message signed by D, A first locates the node containing D's identity and retrieves the associated public key. Next A can discover if there exists a path from itself to D's node in the trust graph. A path in a trust graph is formed by first locating the nodes in the path and second by verifying each signature in the path. Because of the structure of the trust graph, each certificate in the path can be verified using the public key contained in the predecessor's certificate. In this example one such path is (A, C, D). This path indicates that A has signed C's certificate, and C has signed D's certificate. If A trusts C, then A can verify C's signature of D's binding using C's public key.

However, A does not have to simply trust C. Device A can continue to look for other paths to D in the trust graph to validate the binding signed by C. In this example, another such path is (A, B, D). Device B signed the same binding that C signed. Device A can verify B's signature of D's binding using B's public key. Because this second path does not contain device C, device A can be confident that device C alone could not have forged the binding for D. More formally, 2 such independent paths are *vertex-disjoint paths*.

Through this path checking strategy, DecentCA is able to validate bindings between

an identity and public key. As seen in this example, at least two independent paths are required in order to validate the correctness of a binding in the face of a single attacker. For this reason, DecentCA requires at least two vertex-disjoint paths between any two nodes.

This assumption is not unreasonable for well designed networks. A biconnected network will provide at least two separate paths between any two nodes. The redundancy added by this property ensures that no single node failure (or attack) can partition the network into two pieces.

4.2 OSPF Neighbor to Neighbor Authentication

DecentCA also depends on the OSPF neighbor to neighbor authentication scheme introduced in Section 1.2. All communications between neighboring routers are validated using this scheme. The following gives a general description of OSPF neighbor to neighbor authentication.

Initially, s and r share a symmetric key K and have agreed upon a one way hash function H. Version 2 of the OSPF protocol specification allows for different hash functions, but completely specifies the protocol for use with MD5 [24].

1. Router s generates the hash of the message m concatenated with the symmetric key K.

$$H(m + K)$$

2. Router s transmits both the message and the computed hash to r.

$$m + H(m + K)$$

3. Upon receipt of this message, router r can verify the message by generating the hash of $m+K$ and comparing it with the hash transmitted in the previous step.

If the two hashes do not match, the message has been tampered with or was not generated by s.

4.3 Decentralized Certificate Authentication

DecentCA can be divided into three stages:

1. Certificate generation and flooding
2. Trust database exchange
3. LSA validation

4.4 Certificate Generation and Flooding

During the certificate generation and flooding stage, a security association is established between neighboring routers. Each router generates a message containing its identity and public key. This message is verified by the neighboring router using its pre-configured knowledge of the originating router's public key. Once verified, the neighboring router digitally signs a certificate containing the originating router's public key and identity. This certificate is flooded to area routers. Pre-configured information need only include each neighbor's public key. This public key is exchanged out of band at the same time as the shared key used in neighbor to neighbor authentication previously discussed.
Given that:

- Router r has public key $public_r$,
- router s can encrypt a message m using its private key,

$$E_s(m)$$

- routers r and l can decrypt the message m with s's public key,

$$D_s(E_s(m)) = m$$

- and routers r, s, and l have agreed upon a one-way hash function for a message m,

$$H(m)$$

certificate generation and flooding proceeds as follows (as illustrated in Figure 4.2):

1. When router r comes on line, it transmits its identity and public key to neighboring router s.

$$r + public_r$$

2. When router s receives this message, it in turn transmits its identity and public key to r.

$$r + public_r$$

3. Router s verifies that $r + public_r$ matches the public key for r received out of band. If it does not, certificate generation is aborted.

4. Router r verifies that $s + public_s$ matches the public key for s received out of band. If it does not, certificate generation is aborted.

5. Router s generates a hash of $r + public_r$ and encrypts it using its private key.

$$E_s(H(r + public_r))$$

6. Router r generates a hash of $s + public_s$ and encrypts it using its private key.

$$E_r(H(s + public_s))$$

7. Router s floods a certificate containing r's identity, r's public key, and s's identity, followed by s's signature throughout the routing area.

$$r + public_r + s + E_s(H(r + public_r))$$

8. Router r floods a certificate containing s's identity, s's public key, and r's identity, followed by r's signature throughout the routing area.

$$s + public_s + r + E_r(H(s + public_s))$$

9. Router l, being some router in the area not s, receives

$$r + public_r + s + E_s(H(r + public_r))$$

and

$$s + public_s + r + E_r(H(s + public_s))$$

10. Router l inserts the received data into its trust database.

4.5 Trust Database Exchange

At this point in DecentCA, all nodes in the network possess r's public key as well as r's signature of its neighbor's public keys. However, r does not possess public keys for nodes in the network other than its immediate neighbors. The trust database exchange stage ensures that when a new router r comes on line, it receives the existing trust information about the network. During this stage, r receives each neighbor's trust database and adds the new information to its own trust database.

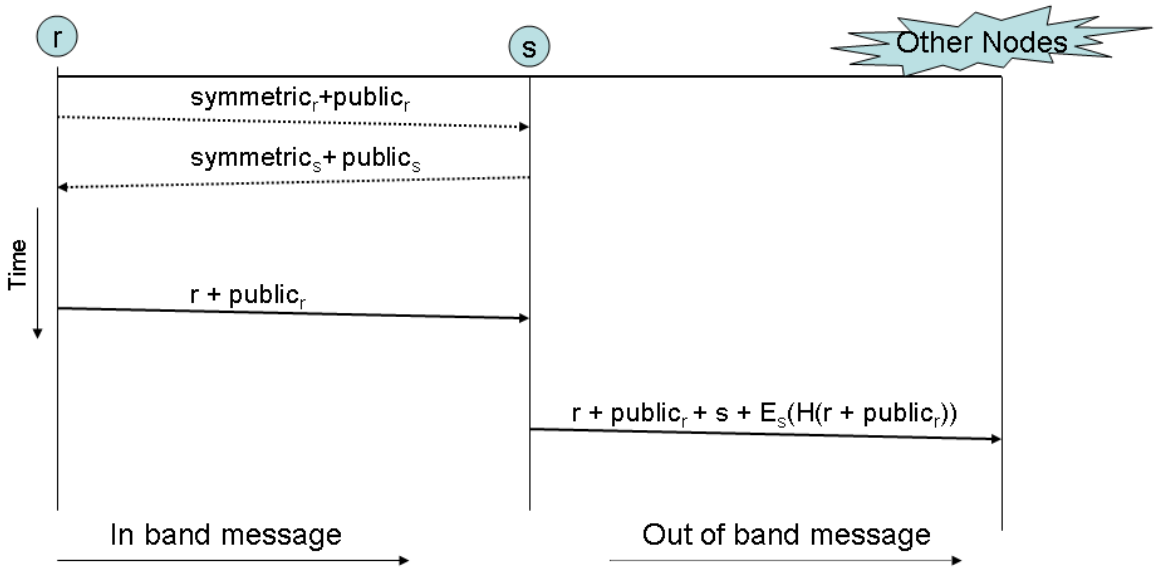


Figure 4.2: The certificate generation and flooding stage of DecentCA.

Given that r has trust database T_r and s has trust database T_s, the exchange proceeds as follows:

1. Router s transmits T_s to r.

2. Router r adds the information from T_s to T_r.

$$T_r := T_r \cup T_s$$

4.6 LSA Validation

When a signed message is received, it includes the signer's identity. The receiver verifies that the signer's public key exists in its trust database, and that there exists a path from its own public key to the signer's public key in the trust graph.
Assuming l and r are routers in network G, and that r has public key $public_r$, verification proceeds as follows:

1. Router l ensures that all neighbors have issued a certificate for l's public key. If

this is not the case, the received LSA is marked as unverified and added to the database. However, unverified LSA's are not used in route calculation. When new certificates are received, an attempt is made to verify all unverified LSA's.

2. When l receives an LSA originating from r, l verifies that

$$l \in V_T$$

and that there exists some path in its T from l to r. If this does not hold, the LSA is flagged as unverified.

3. Router l verifies that there exists no node r with public key $public'_r$ different from $public_r$.

   (a) If this does not hold, l checks for the existence of disjoint paths $p_1, p_2, \ldots, p_n$ for $n > 1$ from l to r with public key $public_r$ in its T.

      i. If this holds, l uses $public_r$ to verify the LSA.

      ii. Otherwise, l uses $public'_r$ to verify the LSA.

LSA validation requires locating a public key with multiple paths from the source node to the destination node in the trust graph. Because certificates are generated for all neighbors in DecentCA, the trust graph takes on the exact topology of the physical network. Given the assumption of a biconnected network, the trust graph must also be biconnected. The first step in LSA verification ensures that all nodes issue certificates for all neighbors. If this is not the case, the affected node cannot verify any LSA's and is non-functional. This prompts immediate action by network administrators.

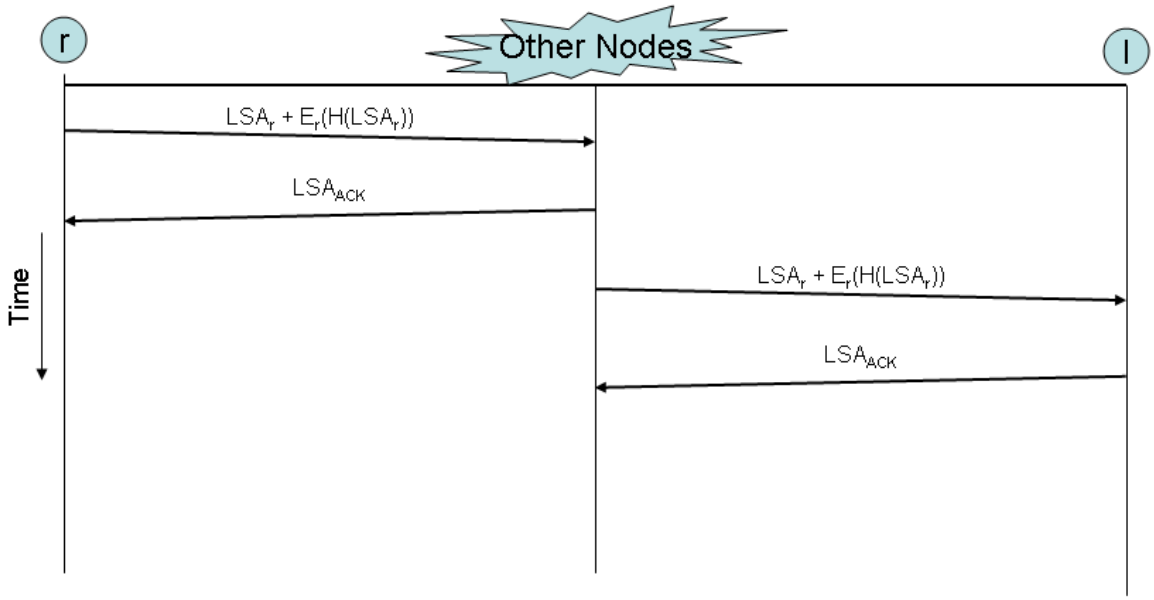


Figure 4.3: The LSA validation stage of DecentCA.

4.7 Certificate Revocation

An important part of any public key infrastructure is certificate revocation. In a standard centralized public key infrastructure, a certificate revocation list (CRL) is generated by the certificate authority. Entities wishing to authenticate certificates query the CRL for the presence of the certificate in question. In a decentralized public key infrastructure where no single certificate authority exists, this approach is not feasible.

A certificate is revoked in DecentCA by each party that originally signed the certificate in question. Each neighbor floods a certificate revocation message. This message includes the identity and public key being revoked signed by the device revoking the certificate. When this message is received by another device, the signature is checked. If the signature is valid, the certificate is removed from the trust database. However, This results in the revoked certificate becoming disconnected from the trust graph.

Chapter 5

Attack Models

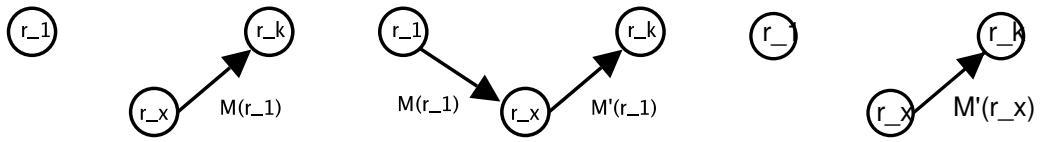
Validation of security protocols is a daunting task. Schneier [26] explains,

> No amount of beta testing can ever uncover a security flaw. The only way to have any confidence in the security of a system is over time, through expert evaluation.

It is with this understanding that this thesis attempts to cover the major attack models discussed in the literature. It is not the intent of this thesis to attempt an exhaustive search of all possible attacks. From Schneier's statement such an undertaking is nearly impossible. It is hoped that this research will prompt the community to examine DecentCA for deficiencies and iteratively improve upon those deficiencies where found.

Elsewhere Schneier discusses threat modeling as one way to build confidence in the security of a system. Three threats, discussed by Huang et al. [**?**], relating to link state routing are considered and illustrated in Figure 5.1:

- **Masquerading:** Impersonating other routers
- **Substitution:** Substituting valid information with falsified information



(a) Masquerade Attack: Router r_x sends a message M, which is purported to have originated from r_1.

(b) Substitution Attack: Router r_x receives a message M from r_1, modifies its contents, and sends the modified message, M', to r_k.

(c) Insertion Attack: r_x originates a message M' containing false information about its attached networks.

Figure 5.1: OSPF attack models.

- **Insertion:** Inserting information that makes false claims about networks to which a router is attached

These attacks may be carried out under various circumstances. The attacker can either be an outsider or an insider. An outsider attack is undertaken by a router that has not been authorized to participate in the routing protocol. Outsider attacks are perpetrated by an individual who gains access to a link incident to authorized routers. OSPF routing packets can then be introduced directly into the network by the perpetrator. In addition, OSPF routing data is sent using the Internet Protocol (IP) and can conceivably originate anywhere on the Internet with a destination address of an authorized router.

An insider attack, on the other hand, is performed by a router that is authorized to participate in the routing protocol but was later subverted. This means the router knows the secret keys needed to perpetuate routing information beyond its immediate neighbors. Insider attacks can further be classified according to the following criteria:

- **Number of attackers:** Attacks can include single or multiple attackers.
- **Relationship among attackers:** Colluding attackers work together to undermine security efforts whereas non-colluding attackers work independently.

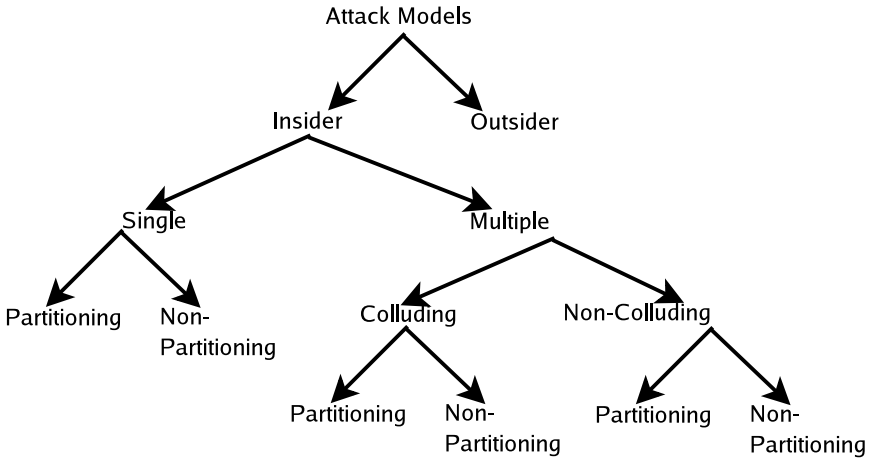


Figure 5.2: OSPF insider attack models.

- **Position:** When attackers partition the network, all traffic crossing from one side of the network to the other must pass through attackers. However, when attackers do not partition the network, there is at least one path that does not pass through a malicious node.

Figure 5.2 summarizes the different combinations of attacks.The remainder of this chapter contains a threat analysis o each attack..

To verify the effectiveness of DecentCA, a threat analysis of each attack model is performed.

5.1 Outsider Attacks

First, all outsider attacks may be considered together. Outsider attacks are thwarted by the OSPF neighbor to neighbor authentication when cryptographic authentication is used. Because the outsider, by definition, is not trusted by any router in the network, it does not know any of the secret keys used by participating routers. Once the forged routing data enters the routing domain, the first router to receive the information cannot verify the message using the scheme introduced in Section 4.2. The message is then discarded. This holds true for all combinations of single or

multiple colluding or non-colluding attackers.

5.2 Insider Attacks

Insider attacks take place when a router becomes subverted or malfunctions. This router is referred to as "malicious" whether it is subverted or malfunctioning. The malicious router may then introduce incorrect information into the routing domain. When no security measures are in place, this routing information, correct or incorrect, will be accepted by other routers in the routing area. If the OSPF cryptographic security between neighbors is employed, this information would still be accepted and forwarded because the malicious router has knowledge of the secret key used by its neighbors. Following is a discussion of other attack scenarios illustrated in Figure 5.1.

5.2.1 Single Attacker - Masquerading

A masquerading attack undertaken by a single attacker r_x occurs when r_x floods information purported to originate from a router r_1, not r_x. This information must be signed by r_1 using $private_{r_1}$ in order to be validated by another router r_k, using $public_{r_1}$. However, r_x does not know r_1's private key and cannot sign a message that can be verified using r_1's public key. Router r_x must flood a forged version of r_1's public key, $public'_{r_1}$ for which it knows the private key $private'_{r_1}$. In addition, in order for this forged key to be linked into the trust database, it must be signed by some router already in the database. Because r_x only knows its own private key, it must use this key to sign the forged key.

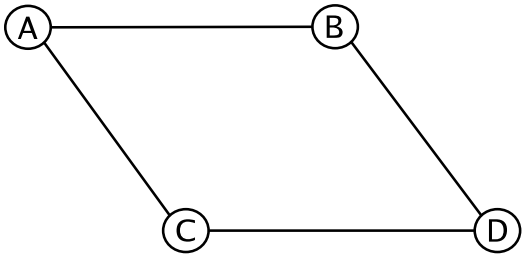
After flooding, each router's trust database contains two different versions of r_1's public key, $public_{r_1}$ and $public'_{r_1}$. If this is not the case, r_1 must not be online, and each router's trust database only contains the forged key. If r_1 is not online, there is no harm accomplished by impersonating it. Given that r_1 is online, in order to

determine which is the correct key, r_k must first verify that there exists a trust path from r_k to r_1. If such a path does not exist containing one of these keys, the missing key can be discarded. However, as mentioned above, r_x signed $public'_{r_1}$ and assuming r_x was already trusted (otherwise it would not be in the trust database) there does exist a trust path from r_k to r_1 through r_x that contains $public'_{r_1}$. In addition, because r_1 has already flooded its public key, there exists a trust path from r_k to r_1 containing $public_{r_1}$.

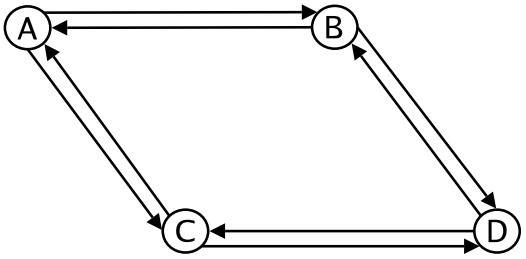
In this scenario, two keys exist, both of which appear to be trusted. At this point, two cases must be considered. If r_x does not partition the network, there exist at least two paths between r_k and r_1. If so, there will also be at least two paths between r_k and r_1 containing $public_{r_1}$ in the trust graph, given the assumption of at least two independent paths between any two nodes. However, because r_x does not know any other router's private keys, there can only be a single trust path between r_k and r_1 containing $public'_{r_1}$. The existence of multiple paths containing one key, and only a single path containing the other, uncovers the fact that $public'_{r_1}$ must be forged.

In the second case r_x partitions the network between r_k and r_1. In this case there only exists a single path from r_k to r_1 and that path contains r_x. This means that all traffic from r_k to r_1 must pass through r_x, and r_x can corrupt all traffic between r_1 and r_k no matter what security measures are in place.

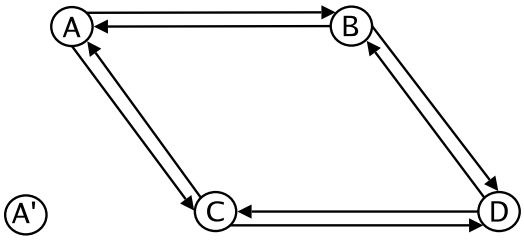
For example, in Figure 5.3(a) C wishes to masquerade as A to D. To begin with all public keys have been exchanged. Figure 5.3(b) represents the trust graph as seen by node D. Each node in the graph is a certificate for the labeled node. Each edge in the graph is a signature of the destination node's certificate by the source node. Node D therefore possesses the correct public key for A. Node C could simply send a message purporting to originate from A. However, this message must be signed. Because C does not know A's public key, it cannot generate the correct digital signature for a message from A.



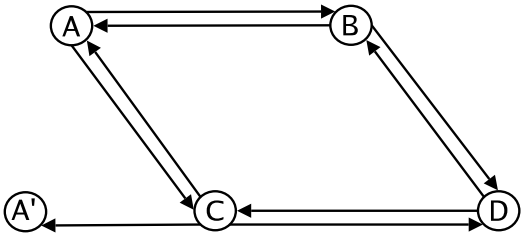
(a) Example network topology.



(b) Node D's initial trust graph.



(c) Node D's trust graph when C floods a forged version of A's public key without signing it. Node A' represents the certificate containing the forged key. It is not linked into D's trust graph because it has not been signed by any nodes in D's trust graph.



(d) Node D's trust graph when C floods a forged version of A's public key signed by C. Node A', the forged certificate is now linked into D's trust graph because it was signed by C.

Figure 5.3: Single masquerading attacker example.

In order to be believed, C must forge a public/private key pair to D. To do this, node C floods a new public key for A for which it knows the associated private key. If C simply floods a new public key for A, the trust graph as seen by D would appear as in Figure 5.3(c). Node A' represents the forged public key for A. Because it has not been digitally signed by any nodes, it is disconnected from D's trust graph. If D ever receives a message signed by the certificate for A', it will not find a path to A' in its trust graph. As a result, D cannot validate the message.

Router C is therefore forced to sign A' with its own private key in order to link the forged public key for A into D's trust graph. Router D trust graph appears as shown in Figure 5.3(d). When D receives a message signed by C using the forged public key for A, it can locate a path from itself to A' in its trust graph. Note however, that it will also find the path to the correct public key for A. In fact, there are two paths to the correct public key for A: one path through B, and one path through C. Because this is assumed to be a biconnected network, D knows that the correct key will be reachable by at least two paths in the trust graph. In this way, D is able to decide that the correct key is the key contained in the certificate represented by node A and not the certificate represented by node A'.

5.2.2 Single Attacker - Substitution

In a substitution attack, a single attacker r_x attempts to modify, and continue flooding modified information originating from r_1. Because the information being flooded has previously been signed using r_1's private key, r_x must generate a new signature for the modified information. However, because r_x does not know r_1's private key, it cannot generate a signature that can be validated using $public_{r_1}$. Therefore, it must flood a forged version of r_1's public key $public'_{r_1}$ for which it knows the corresponding private key $private'_{r_1}$. At this point, the attack is the same as the masquerading attack in Section 5.2.1.

5.2.3 Single Attacker - Insertion

An insertion attack can take on one of two forms. First, a single attacker r_x can insert information about a link, which does not exist, to another node r_1. This attack is undertaken in order to re-route traffic destined for r_1 through r_x. The goal of this attack is to eavesdrop on these conversations or to drop the data with the intent of disrupting communication.

Suppose r_x advertises a link from r_x to r_1 signed using $public_x$. This advertisement is signed by a trusted key and will be accepted as valid. This creates a uni-directional link from r_x to r_1. Because the link does not really exist, r_1 would not generate an advertisement for this link, creating a bi-directional link in the area routers' link state databases. As a result, this link remains a uni-directional link in the area routers' link state databases. The OSPF specification states that only bi-directional links are included in routing calculations, and no computed route in OSPF includes the supposed link between r_x and r_1.

Further suppose that to overcome this problem, r_x generates an advertisement for the link from r_1 to r_x purported to have originated from r_1. This is exactly the scenario presented in Section 5.2.1 and can be overcome by the technique presented there.

5.2.4 Multiple Independent Attackers

When multiple malicious routers exist and are working independently, all three attacks, masquerading, substitution, and insertion essentially become multiple instances of the individual attacks discussed above. This protocol addresses these problems as discussed above.

5.2.5 Multiple Colluding Attackers - Masquerading

If the multiple malicious routers are colluding, the masquerading attack proceeds as follows: routers $r_{x_1}, r_{x_2}, \ldots, r_{x_i}$ generate link state advertisements purported to originate from r_1. Again, like the single attacker version, the advertisement that supposedly originates from r_1 must be signed with $private_{r_1}$. Routers $r_{x_1}, r_{x_2}, \ldots, r_{x_i}$ do not know $private_{r_1}$, so they must forge a public key $public'_{r_1}$ for which they know the corresponding private key, $private'_{r_1}$. The forged key is flooded to the routers in the OSPF area. Each of the malicious nodes could generate, sign, and flood a certificate for this public key. This creates multiple trust paths from r_k to r_1 containing $public'_{r_1}$ in the area routers' trust databases since each of the r_x's is trusted (otherwise they would be outsiders).

After this flooding, each r_k has two different versions of r_1's public key, $public_{r_1}$ and $public'_{r_1}$. Because multiple attackers have signed certificates for $public'_{r_1}$, the decision of which key to accept becomes impossible. However, it is clear that the attack is detected due to the presence of two different public keys for the same identity. In addition, r_k can narrow down the attacking routers to either the group that signed the certificate for $public'_{r_1}$ $(r_{x_1}, r_{x_2}, \ldots, r_{x_i})$ or all routers that signed the certificate for $public_{r_1}$. Furthermore, if $r_{x_1}, r_{x_2}, \ldots, r_{x_i}$ do not partition the network, r_1 will receive the certificate flooding for $public'_{r_1}$ from all of the attacking routers. In this way, r_1, the victim router, can determine the exact identity of the attackers. This information is invaluable to the administrator for r_1 or for an intrusion detection system as a tool to correct the problem.

5.2.6 Multiple Colluding Attackers - Substitution

For a substitution attack involving multiple, colluding attackers, routers $r_{x_1}, r_{x_2}, \ldots, r_{x_i}$, attempt to modify and continue flooding falsified information originating from a single router, r_1. Again as in the single case, a new signature must be generated for

the modified information. Since $r_{x_1}, r_{x_2}, \ldots, r_{x_i}$ do not know the private key for r_1, $private_{r_1}$, they must originate a new public key for r_1, $public'_{r_1}$. From this point on, the attack is exactly the same as discussed in Section 5.2.5, and the analysis performed for that case applies.

5.2.7 Multiple Colluding Attackers - Insertion

An insertion attack undertaken by multiple colluding routers takes on the following form: routers $r_{x_1}, r_{x_2}, \ldots, r_{x_k}, \ldots, r_{x_i}$ collude to insert information about a link to another node r_1, that does not exist in order to draw traffic through any one of the r_{x_k}'s. As in the single attacker scenario, such an advertisement is signed by r_{x_k}'s private key and found to be valid. However, since the link does not really exist, r_1 does not generate an advertisement for this forged link again resulting in a uni-directional link in the link state database of the area routers. This uni-directional link is not used in route calculations, and the attack is ineffective.

Suppose, however, that in order to overcome this, routers $r_{x_1}, r_{x_2}, \ldots, r_{x_k}, \ldots, r_{x_i}$ flood a forged version of r_1's public key, $public'_{r_1}$. From this point on, the attack and analysis is exactly the same as that found in Section 5.2.5.

Chapter 6

DecentCA Implementation

This research focuses on implementing DecentCA in the Quagga routing software suite (http://www.quagga.net) using OpenSSL (http://www.openssl.org) to implement the cryptographic primitives. Quagga is an open source implementation of a number of different routing protocols, including OSPF. It has been available for a number of years and is interoperable with most major commercial routing software. OpenSSL is a standard cryptographic library implementing a number of cryptographic primitives.

In order to map the three sections of the DecentCA algorithm, certificate generation and flooding, trust database exchange, and certificate verification, into an existing software system, this implementation takes advantage of the opaque LSA extension to OSPF [5]. Opaque LSA's allow routers to send arbitrary data to all other routers in the autonomous system, OSPF routing area, or connected to a given link. Opaque LSA's work by encapsulating data being transmitted in one of three reserved LSA types (determined by the flooding scope). Routers advertise their ability to accept such LSA's to their neighbors during the initialization phase of OSPF. Opaque LSA's are flooded, based on the scope specified, to all neighbors advertising such a capability.

6.1 Certificate Generation and Flooding

The certificate generation and flooding section of the DecentCA algorithm is implemented as an extension to the normal OSPF initialization phase. During initialization, each router contacts a router on each link to which it has a connection. A handshake procedure takes place with the responding router to determine configuration compatibility and to agree on certain variable configuration items. Immediately after completion of the handshake, the DecentCA initial handshake is performed. At this point, public keys are exchanged, verified, and flooded as describe in Section 4.4. Initial exchange of public keys is accomplished using link scoped opaque LSA's. Flooding of signed certificates is accomplished by encapsulating the signed certificate in an OSPF area scoped opaque LSA. These opaque LSA's are occasionally refreshed, per standard OSPF configuration parameters, to keep them in the OSPF database of each router.

When each router receives a new opaque LSA, it is processed just as other LSA's. This means that each opaque LSA is stored in the router's link state database for a period of time. This period of time is configured as the MaxAge parameter in OSPF. Each router must refresh the certificates it generates in order to keep the opaque LSA's containing the certificate from "aging out" of the OSPF routing domain. This is done by flooding another copy of the LSA where the sequence number has been incremented.

Having opaque LSA's containing certificates stored in the link state database means that the DecentCA implementation need not keep any persistent storage of received certificates. Each time a received LSA needs to be verified, the trust graph is built from the certificates contained in opaque LSA's stored in the router's link state database.

6.2 Trust Database Exchange

Because certificates are contained in opaque LSA's and stored in the link state database by each router, the trust database exchange is trivial. In standard OSPF implementations, neighbors synchronize their database during the initialization process. At this time, the router sends a summary of its link state database to its neighbor and visa versa. When a router receives this summary, it examines its database and requests any missing LSA's from its neighbor. After initialization is completed, each router has exactly the same database as its. The database includes any opaque LSA's containing certificates.

6.3 LSA Signing and Verification

Each LSA is contained in a link state update packet that contains a list of link state updates. To minimize the impact on existing code, this research proposes that signatures for each LSA be placed in a list immediately following the list of LSA's. In this way, each LSA may be processed as before. Signature verification then takes place by reading each signature in turn from the incoming packet and associating it with the LSA previously processed. The order in which the signatures appear determines which LSA it is associated with.

After signatures are associated with the appropriate LSA, verification takes place. The trust graph is first created from the opaque LSA's contained in the router's trust database. Once the trust database is generated, verification continues as described in Section 4.6.

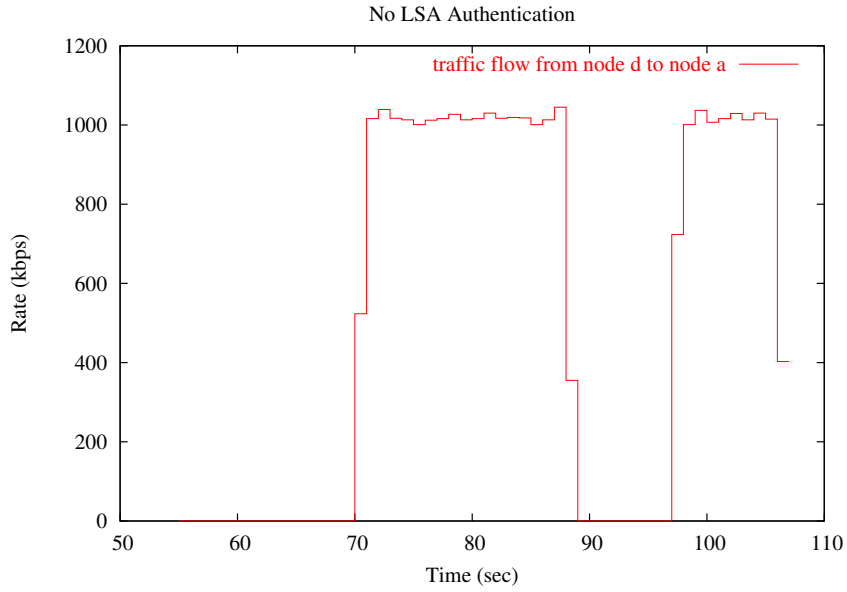


Figure 6.1: An example attack against standard OSPF with no security.

6.4 Example Attack

In addition to standard use case tests, this research implements the attack scenario discussed in Section 5.2.1. Initially routing takes place as usual, and traffic is flowing from D to A through B. However, 60 seconds into the scenario, node C injects a forged public key for node B into the routing domain. Next, C injects a forged LSA from B indicating an increase in cost for the link from B to A. This causes D to recompute the route to A such that it passes through C. Node C, being malicious, drops all traffic flowing from D to A.

In one experimental scenario, no LSA authentication takes place. This represents OSPF as it is currently implemented. In this scenario, Figure 6.1 shows a total loss of communication between 88 and 98 seconds. Communication is restored due to the OSPF fightback mechanism. However, a persistent attacker can automatically continue this attack, effectively implementing a fightback to the fightback. This causes great instability in the network.

In the second experimental scenario, DecentCA is introduced and LSA authentication

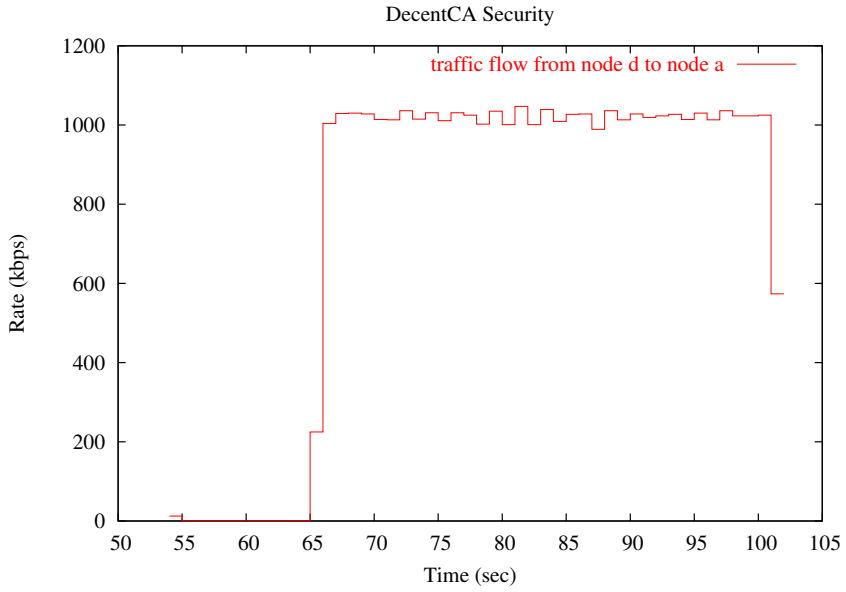


Figure 6.2: An example attack against DecentCA security.

takes place. Figure 6.2 shows that there is no disruption in traffic during the time of the attack. DecentCA correctly identifies and ignores the forged public key and LSA.

6.5 Computational Overhead

When compared with OSPF with digital signatures, the additional computational overhead of the DecentCA protocol includes computing the number of vertex-disjoint paths in the trust graph from the current node to the public key certificate being used to verify signatures (described in Section 4.6). Computing the number of vertex disjoint paths from node s to node t in the trust graph is accomplished using the Ford-Fulkerson method in the Edmonds-Karp algorithm [6]. The goal of the Edmonds-Karp algorithm is to find the max flow from node s to node t. By setting the weight of each link to 1 and applying the max-flow min-cut theorem, the min-cut for the graph is found. However, the min-cut is equivalent to the number of edge-disjoint paths and not the number of vertex disjoint paths. In order to find the number of vertex disjoint paths, the graph is transformed in the following manner before applying the

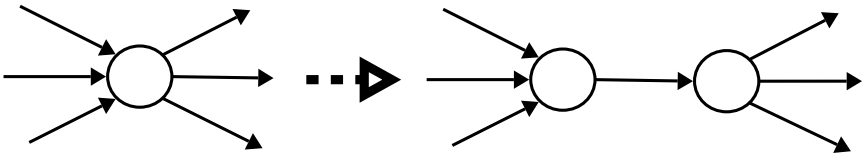


Figure 6.3: Each vertex in the graph is transformed to provide a link that represents each vertex in the graph. When the min-cut of the transformed graph is found, it will represent the maximum number of vertex disjoint paths.

Edmonds-Karp algorithm [23]:

Given a graph D with verticies $V(D)$ and edges $E(D)$, the transformed graph $D\prime$ is constructed as:

$V(D\prime) = \{v_{in}|v \in V(D)\} \cup \{v_{out}|v \in V(D)\}$

$E(D\prime) = \{v_{in}v_{out}|v \in V(D)\} \cup \{u_{out}v_{in}|uv \in E(D)\}$

Figure 6.3 represents this graphically [8].

Each vertex in the graph is transformed to provide a link that represents each vertex in the graph. When the min-cut of the transformed graph is found, it represents the maximum number of vertex disjoint paths.

Cormen et al. [6] show that the complexity of the Edmonds-Karp algorithm is $O(VE^2)$. While this is slightly worse than the most complex operation traditionally performed in OSPF (the shortest path calculation via Dijkstra's algorithm, which is $O(V^2)$), it still runs in polynomial time.

6.6 Network Communication Overhead

The communication overhead in DecentCA, when compared with OSPF digital signatures, results from flooding neighbor signed certificates. In OSPF digital signatures, a certificate signed by the CA are flooded by the subject mentioned in the certificate. However, in DecentCA, a certificate signed by each neighbor must be flooded. This increases the total number of certificates flooded by a factor determined by the degree

of the nodes in the network. For instance, if the network has an average node degree of 3, there will be 3 times the number of certificates flooded.

In this implementation of DecentCA, public keys are transferred in PEM format. PEM is a plaintext version of the public key including a base64 encoded public key, a header, and footer. For a 1024 bit key, the PEM "file" requires 251 bytes with a 16 byte signature by the authorizing (neighbor) router. Each link requires 2 certificates, resulting in an initialization overhead of 534 bytes per link. For a 100 Mbps link, this is less than 5 percent of the total available bandwidth per link. In addition each LSA requires an additional 16 byte signature. With a standard refresh time of around 1 hour and a randomization factor that staggers the refresh time of LSA's, this overhead is insignificant.

Chapter 7

Security Comparison

Table 7.1 compares three security technologies for OSPF. This first is simple cryptographic authentication. This approach authenticates messages between neighbors. If an outside attacker does not have know any router's shared key, all outsider messages will be rejected by the first router to receive the message. However, if the attacker has possession of the symmetric key of one of its neighbors (i.e., it is an insider) cryptographic authentication offers no protection.

Countermeasures	Attack Models						
	Insider						Outsider
	Single		Multiple				
			Colluding		Non-Colluding		
	Partitioning	Non Partitioning	Partitioning	Non Partitioning	Partitioning	Non Partitioning	
Cryptographic Authentication							✓
Digital Signatures	✠	✓	✠	✓	✠	✓	✓
DecentCA	✠	✓	⚹	⚹	✠	✓	✓

✓: Attack is overcome
✠: Attack is overcome in individual partitions but not between partitions
⚹: Attack is detected

Table 7.1: Characteristics of different OSPF security proposals.

Digital signatures provide the highest level of security of the three. This is mainly due to its resilience to multiple, colluding attackers. In the digital signatures approach, a third party, the certificate authority, is responsible for binding each router's identity to its public key and signing a certificate asserting the binding. The difficulty with this approach is that the single certificate authority offers attackers a single point of vulnerability. By subverting the certificate authority, all security in the network is compromised. A subverted certificate authority can be used to revoke previously issued certificates and issue new certificates for which the attacker know the associated private keys.

DecentCA provides a compromise between the digital signatures approach and the simple cryptographic authentication approach. While DecentCA does not provide the full security of a centralized CA, it does offer security for the attacks that are most likely to occur. Subverting a single router can be fairly simple, but subverting multiple routers may prove much more complicated. In addition, attacks that are not entirely protected against (such as those by multiple colluding attackers or attackers that partition the network) are either detected or partially defended against. For example, when an attacker can partition the network, communications between routers in the same partition remains secure.

Chapter 8

Conclusion

This research presents an algorithm for distributing public keys used in the OSPF digital signatures infrastructure. The proposed algorithm, DecentCA, provides a decentralized certificate authentication system. Initially, public keys are certified by neighboring routers. These certificates are flooded through the network and stored. As part of the certificate chain verification, vertex disjoint paths in the trust database are used to verify the validity of certificates. Link state updates are signed by the originating routers and verified using the flooded certificates. In addition, an implementation of DecentCA is created using the Quagga routing software suite and OpenSSL. This demonstrates the feasibility of implementing DecentCA. Further, example attacks are performed, and the results of one of these attacks is presented.

DecentCA is novel as it does not rely on a central certificate authority to authenticate router certificates, thus overcoming the need to administer a central certificate authority. Second, it removes the single point of failure and attack present when a central certificate authority is used. DecentCA is shown to be robust in the face of single or multiple non-colluding attackers. In addition it is shown to detect attacks in the face of multiple colluding attackers. While not quite as strong as using a central certificate authority, the author believes that the benefits of decentralizing the

security infrastructure and lowering the administrative overhead involved offset this weakness.

The advantage of DecentCA over the existing OSPF neighbor to neighbor security is clear from Table 7.1. DecentCA offers increased security against insider attacks at a minimal increase in administrative overhead. OSPF Neighbor to Neighbor security requires the exchange of a symmetric key, and DecentCA requires the exchange of public keys at the same time. By offering this increase in security with minimal increase in administrative overhead, these security measures are easily adopted. In this way, the network as a whole becomes significantly more secure.

Bibliography

[1] Foundry Enterprise Configuration and Management Guide, 2004, http://www.foundrynet.com/services/documentation/ecmg/OSPF.html.

[2] Quagga Routing Software Suite, Apr 2004, http://quagga.net/.

[3] R. Albert, H. Jeong, and A. Barabási. The Internet's Achilles' Heel: Error and Attack Tolerance of Complex Networks. *Nature*, 406:378–382, 2000.

[4] A. Chakrabarti and G. Manimaran. Internet Infrastructure Security: A Taxonomy. *IEEE Network*, 16(6):13–21, Nov/Dec 2002.

[5] R. Coltun. The OSPF Opaque LSA Option. RFC 2370, Jul 1998, http://www.faqs.org/rfcs/rfc2370.html.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algortihms*. McGraw Hill, New York, New York, USA, 1990.

[7] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov 1976.

[8] M. X. Goemans. Topics in Combinatorial Optimization: Lecture 22. MIT OpenCourseware, May 2004.

[9] M. T. Goodrich. Efficient and Secure Network Routing Algorithms, 2001, http://www.cs.jhu.edu/ goodrich/cgc/pubs/routing.pdf.

[10] R. Hauser, T. Przygienda, and G. Tsudik. Lowering Security Overhead in Link State Routing. *Computer Networks*, 31(8):885–894, 1999.

[11] D. Huang, A. Sinha, and D. Medhi. A Double Authentication Scheme To Detect Impersonation Attack In Link State Routing Protocols. In *Proceedings of the IEEE International Conference on Communications*, pages 1723–1727, Anchorage, Alaska, USA, May 2003.

[12] B. Kaliski and J. Staddon. PKCS #1: RSA Cryptography Specifications Version 2.0. RFC 2437, Oct 1998, http://www.faqs.org/rfcs/rfc2437.html.

[13] L. Lamport. Password Authentication With Insecure Communication. *Communications of the ACM*, 24(11):770–772, Nov 1981.

[14] P. McDaniel and S. Jamin. A Scalable Key Distribution Hierarchy. Technical Report CSE-TR-366-98, Jul 1998.

[15] J. Moy. OSPF Version 2. RFC 2328, Apr 1998, http://www.faqs.org/rfcs/rfc2328.html.

[16] S. Murphy, M. Badger, and B. Wellington. OSPF With Digital Signatures. RFC 2154, Jun 1997, http://www.faqs.org/rfcs/rfc2154.html.

[17] S. Murphy, R. Mundy, O. Gudmundsson, and B. Wellington. Retrofitting Security into Internet Infrastructure Protocols. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, volume 1, pages 3–17, Hilton Head, South Carolina, USA, Jan 2000.

[18] National Bureau of Standards and Technology. Data Encryption Standard (DES), Dec 1993, http://www.itl.nist.gov/fipspubs/fip46-2.htm.

[19] National Institue of Standards and Technology. Data Signature Standard (DSS), May 1994, http://www.itl.nist.gov/fipspubs/fip186.htm.

[20] National Institute of Standards and Technology. Advanced Encryption Standard (AES), Nov 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[21] R. Perlman. *Network Layer Protocols With Byzantine Robustness*. PhD thesis, Massachusetts Institute of Technology, August 1988.

[22] R. Perlman. An Overview of PKI Trust Models. *IEEE Network*, 13(6):38–43, Nov/Dec 1999.

[23] B. Reed. Finding Disjoint Paths, http://cgm.cs.mcgill.ca/ breed/308362B/disjpath.ps.

[24] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, Apr 1992, http://www.faqs.org/rfcs/rfc1321.html.

[25] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, New York, New York, USA, second edition, 1996.

[26] B. Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, New York, New York, USA, 2000.

[27] S. F. Shamim. OSPF: Frequently Asked Questions, Nov 2004, http://www.cisco.com/warp/public/104/9.html.

[28] W. Stallings. *Network Security Essentials: Applications and Standards*. Prentice Hall, Upper Saddle River, New Jersey, USA, second edition, 2003.

[29] B. Vetter, F. Wang, and S.F. Wu. An Experimental Study of Insider Attacks for OSPF Routing Protocol. In *Proceedings of the IEEE International Conference on Network Protocols*, pages 293–300, Atlanta, Georgia, USA, Oct 1997.

[30] M. Yu. A Secure Routing Protocol With Limited Key Distribution and Double Message Hashing. In *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, volume 2, pages 553–556, Victoria, British Columbia, Canada, Aug 2003.

[31] L. Zhou and Z. J. Haas. Securing Ad-Hoc Networks. *IEEE Network*, 13(6):24–30, Nov/Dec 1999.

[32] P. Zimmermann. *The Official PGP User's Guide*. The MIT Press, Cambridge, Massachusetts, May 1995.