The Dissertation Committee for Eunjin Jung

certifies that this is the approved version of the following dissertation:

Dispersability and Vulnerability Analysis of Certificate Systems

Committee:

Mohamed G. Gouda, Supervisor

Lorenzo Alvisi

James C. Browne

Ehab S. Elmallah

Aloysius K. Mok

Vitaly Shmatikov

Dispersability and Vulnerability Analysis of Certificate Systems

by

Eunjin Jung, B.S.; M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2006

To my parents, Byungkun Chung and Hyesook Koo, who always believe in me

Acknowledgments

I would like to thank my advisor Dr. Mohamed G. Gouda for his support and guidance throughout my Ph.D. program. He led and taught me through not only in research but also in life. I am very grateful to all my committee members for their time and attention, especially to Dr. Lorenzo Alvisi for advice in critical times, to Dr. James C. Browne for intuition and insights, to Dr. Ehab S. Elmallah for wonderful and fruitful collaborations, to Dr. Aloysius K. Mok for continuous encouragement, and to Dr. Vitaly Shmatikov for many interesting discussions. I also would like to thank Dr. Kathryn S. McKinley for the great teaching experience and generous support.

I cannot thank enough all my friends, both in Korea and in the States. Without them I would not be where I am now. Especially Jiwon, Anna, and Jean-Philippe, I owe them so many days of happiness.

Last but not least, I would like to thank my family just for being my family. They always believed in me, and loved me, and I love them very much.

Eunjin Jung

The University of Texas at Austin

August 2006

Dispersability and Vulnerability Analysis of Certificate Systems

Publication No. _____

Eunjin Jung, Ph.D.
The University of Texas at Austin, 2006

Supervisor: Mohamed G. Gouda

A certificate is a way to distribute public keys of users in a distributed system. For example, in the current Internet, certificates are heavily used in SSL/TLS for securing e-commerce. In this thesis, we describe the three phases of a certificate, how a certificate is issued, used, and revoked/expired. In particular, we propose a new way of distributing certificates, called *certificate dispersal*. Certificate dispersal assigns certificates to users such that when a user u wants to securely communicate with another user v in a system, users u and v may find out the public key of user v based on the certificates stored in u or v. In other words, users u and v have no need to contact any other user in the system. We define dispersal in two

environments, a certificate graph and a certificate chain set and the costs of dispersal. In the environment of certificate chain set, computing an optimal dispersal is NP-complete. However, we identify several classes of chain sets and certificate graphs for which optimal dispersal can be computed in polynomial-time. For each class we present an algorithm that computes an optimal dispersal. We also analyze the *vulnerability* of certificate systems. Any certificate system suffer from impersonation attacks when a private key of a user is revealed to an adversary. We define the metric called vulnerability that measures the scope of damage when some private keys are revealed, and show how different certificate systems have different vulnerabilities. These results can be used to design a good certificate system that satisfies system requirements of dispersal cost and vulnerability.

Contents

Chapter 1

Introduction

The concept of public key cryptography (also known as asymmetric key cryptography) was first introduced by Diffie and Hellman in [14]. The main idea is that two keys work as a pair: one that is known to the public and the other that is only known to one user. If a message is encrypted with one key, then the encrypted message can be only decrypted by the other key.

If a user u encrypts a nonce with the public key of another user v and then receives the same nonce from user v, then user u can be assured that user v owns the corresponding private key and authenticate user v. If a user u sends a message to another user v with the hash of this message encrypted with the private key of u, and then user v computes the same hash of the received message as the decrypted hash with the public key of user u, then user v can be assured that user u generated this message. This encrypted hash of a message is called digital signature.

Many distributed protocols and security protocols, both in research and practice, utilizes authentication and digital signature provided by public key cryptography. For example, e-commerce on the Internet operates on Secure Socket Layer (SSL), and SSL authenticates websites using public key cryptography. In the distributed computing literature, some replicated state machine protocols [10], use

public key cryptography, as well as some quorum protocols [29].

Protocols that use authentication and digital signatures based on public key cryptography require an underlying public key infrastructure (PKI). PKI includes the issuance, distribution, and revocation of public keys: how to issue the public and private key for users, how to distribute (and store) the public keys of users, and how to revoke the public keys when the corresponding private keys are revealed to adversaries. In one of the simplest form of PKI, every user stores the public keys of all other users. This PKI does not scale very well when the number of users grows to millions, as in the Internet. Instead, SSL relies on a handful of public keys that are well-known to each user to introduce more public keys. This introduction is done by a digitally signed statement, which is called a "certificate".

A certificate can be understood as an electronic identification. A traveler who wants to pass the security check at any airport in the States needs to provide a photo identification along with a boarding pass. The name on the boarding pass needs to match the name on the photo identification, and the photo in the identification must match the traveler. In other words, the photo identification states the relationship between the name and the face of the traveler.

In the context of PKI, a certificate states a relationship between a user and its corresponding public key, and it is signed by a private key of another user. We call the user who signed the certificate with its private key the "issuer" and the user whose public key is stated in the certificate the "subject" of this certificate. Any user who knows the public key of the issuer can verify the certificate to make sure that it is indeed signed by the issuer. When the certificate is successfully verified, than the user may use the key written in the certificate as the public key of the subject. In other words, the issuer introduces the public key of the subject to this user.

In SSL, most web browsers have public keys of Certificate Authorities (CA).

When a client wants to authenticate a website, then the website provides a certificate to the client that is signed by a well-known CA. The client uses the public key of the CA in the web browser to verify the certificate, and then uses the public key in the certificate for the subsequent authentication protocol. In other words, the CA introduces the public key of the website to the client. In the example of a traveler at the airport, a government (or the agency that issues a state identification) is a CA. A security personnel can trust the government and trust the photo in the identification to be the correct photo for the name. A client can trust the CA and trust that the public key in the certificate is the correct public key for the website.

Certificates in PKI help with scalability. Once the government issues a photo identification, a traveler may use the identification for many trips. Similarly, once the CA issues a certificate for a website, a website may use the certificate for many clients. In the example of SSL, a handful of public keys are stored in each web browser, and millions of clients can authenticate thousands of websites with verifiable certificates.

Users may use more than one certificate to learn public keys of other users. For example, if a user u issues a certificate for another user v and user v issues a certificate for a user w, then user u can learn the public key of user w using the two certificates issued by users u and v. Pretty Good Privacy (PGP) is an example system where more than one certificate can be used. The series of certificates is called a certificate *chain*.

In a distributed system, when a user u wants to find the public key of user v, user u may need to use more than one certificate. If there is a central repository of certificates, user u may query the repository for certificates. However, it is hard to maintain such a repository for a large scale distributed system. In particular, if that system operates on an ad-hoc network, the reachability of the certificate repository becomes non-trivial. We propose a novel way of distributing certificates so that the

users u and v can find all the necessary certificates without contacting any other user. We call this distribution mechanism *certificate dispersal*. Certificate dispersal minimizes the communication overhead in finding certificates. We prove that computing a certificate dispersal that minimizes the average number of certificates stored in a user is NP-Complete in general. The certificate dispersal is optimal if the average number of certificates stored in a user is minimum. We identify several classes of certificate systems and present algorithms that compute optimal certificate dispersals for such systems in polynomial time.

A certificate system may suffer from impersonation attacks. An impersonation attack occurs when an adversary gets hold of the private key of a user u, and pretends to be user u by decrypting messages encrypted with the public key of user u. The adversary can also impersonate another user v using the private key of user u as follows. The adversary may create a new public and private key pair, and issue a certificate with this new public key as if this public key belonged to user v. When a user w is not aware that the private key of user u is revealed to the adversary, user w may use the certificate issued by the adversary and learn the wrong public key. When user w sends any message to user v that is encrypted with the wrong public key, the adversary can intercept the message and learn its content.

In Chapter 4, we define a metric called "vulnerability" of a certificate system which measure the potential damage from impersonation attacks. We also identify what properties of the certificate system affect vulnerability. The analysis of the interaction between these properties gives guidelines on designing a good certificate system.

A certificate has a lifetime in a PKI. It is created by the issuer, is used by users who know the public key of the issuer, and dies when it is revoked or expires. In the next chapter, we define a certificate and a certificate system more formally in the order of these events. In Chapter 3.1, we define certificate dispersal and

explain how a dispersal can be computed for any certificate system, and how to compute an optimal dispersal for some classes of certificate systems. In Chapter 4, we continue by analyzing potential vulnerabilities of certificate systems. Finally we discuss related work and conclude with future directions.

Chapter 2

Certificates and Certificate Systems

We consider a system where users would like to send messages securely to other users. A user who would like to send a secure message is called a *source* and a user who is intended to receive such a message is called a *destination*.

In the Internet, it is common that one source may wish to send messages to many destinations. For example, a source Alice may wish to send her credit card number securely to several destination shopping sites, say Amazon.com, eBay.com, and priceline.com. The secure communication between a source and a destination is protected by encrypting each exchanged message with a shared key only known to the source and destination.

In this system, each user u, whether source or destination, has a private key rk_u and a public key bk_u. In order for a source u to share a key sk with a destination v, u encrypts key sk using the public key bk_v of v and send the result, denoted $bk_v\{sk\}$, to v. Only v can decrypt this message and obtain key sk shared with u. This scenario necessitates that u knows the public key bk_v of v. In the above example, Alice needs to know the public keys of Amazon, eBay, and priceline.

If a user u knows the public key bk_v of another user v in the network, then u can issue a certificate, called a certificate from u to v, that identifies the public key bk_v of v. This certificate can be used by any user that knows the public key of u to further acquire the public key of v.

A certificate from u to v is of the following form:

$$< u, v, bk_v > rk_u$$

This certificate is signed using the private key rk_u of u, and it includes three items: the identity of the certificate issuer u, the identity of the certificate subject v, and the public key of the certificate subject bk_v. Any user that knows the public key bk_u of u can use bk_u to obtain the public key bk_v of v from the certificate from u to v. Note that when a user obtains the public key bk_v of user v from the certificate, the user not only finds out what bk_v is, but also acquires a proof that bk_v is indeed the public key of user v.

A certificate has a lifetime. The issuer issues this certificate, users use this certificate to find the public key of the subject, and the issuer may revoke this certificate or the certificate may expire. We will discuss the first two phases in more details below. (The revocation step will be discussed in Section 3.6.)

2.1 Certificate Issuance

To issue a certificate $< u, v, bk_v > rk_u$, the issuer u must take the following three steps.

i. Find the public key bk_v of user v: In this step, user u needs to make sure that the key bk_v is the correct public key of user v.

ii. Compute the hash (message digest): In this step, user u assembles all the information that will be included in the certificate, in addition to the identity

of the issuer u, the identity of the subject v, and the public key bk_v of user v. For example, the certificate may include the expiration date of this certificate, and the name of the hash function that is used to compute this hash.

iii. Encrypt the computed hash with the private key rk_u: In this step, user u encrypts the computed hash with the private key so that any user who knows the public key of user u can verify that no adversary tampered with this certificate.

Among these three steps the hardest step is the first one, to find the public key of another user in the system. Two different types of users may take this challenge.

i. Certificate Authority: A Certificate Authority (CA) in a system finds the correct public key of other users and issues certificates for them. In some cases, a CA even generates the public and private key for other users and assigns private keys to them. There can be multiple CAs in the system, and each user may have multiple CAs issue certificates for the same public key.

ii. Any user: Any user in a system finds the correct public key of other users and issues certificates for them. Most times such finding relies on verification on an offline channel, for example social contacts. A user may issue as many certificates as he or she wishes, so every user may have different number of certificates issued for himself or herself.

Note that finding the correct public key of a user is more than finding the public key that matches a given private key. More importantly, the identity in the certificate issued for a certain public key should match the real owner of the corresponding private key. For example, recently there was a certificate issued for Mountain America Credit Union in Utah by Equifax Secure Inc., which is a division of the well-known credit reporting bureau Equifax, now part of the company

Geotrust. Geotrust currently holds around 25% of the market share in SSL certificate issuance business. It turned out that the public key in this certificate did not belong to the claimed Mountain America credit union, but to an attacker who collected credit card numbers from Mountain America credit union account holders. (The details can be found in [26].) Reiter and Stubblebine [34] noted that there should be offline verification in certificate issuance.

2.2 Certificates to find public keys

The certificates issued by different users in a network can be represented by a directed graph, called the *certificate graph* of the network. Each node in the certificate graph represents a user in the network. Each directed edge from node u to node v in the certificate graph represents a certificate from u to v in the network.

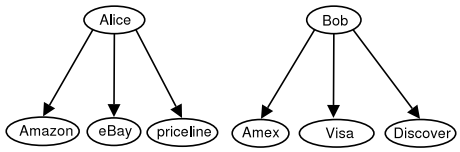


Figure 2.1: A certificate graph of Alice and Bob

Fig. 2.1 shows a certificate graph for a network with two sources, Alice and Bob, and six destinations, Amazon, eBay, priceline, Amex, Visa, and Discover. According to this graph,

Alice issues three certificates

($Alice$, $Amazon$), ($Alice$, $eBay$), and ($Alice$, $priceline$), and

Bob issues three certificates

(Bob, $Amex$),(Bob, $Visa$), and (Bob, $Discover$)

A more efficient way to support secure communication between the sources

and the destinations is to introduce some intermediaries between the sources and the destinations. The number of introduced intermediaries is much smaller than the number of sources and the number of destinations. Each intermediary has its own public and private key pair. The sources know the public keys of intermediaries and the intermediaries issue certificates of the public keys of the destinations. For example, two intermediaries, namely VeriSign and CertPlus, can be introduced between the two sources and the six destinations in Fig. 2.1. The result is the certificate graph in Fig. 2.2.

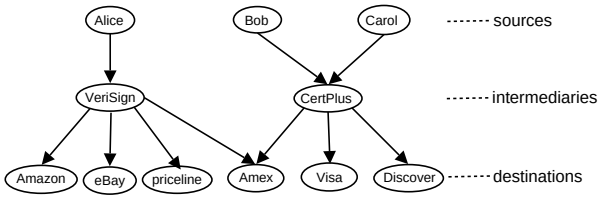


Figure 2.2: A certificate graph with intermediaries

According to the certificate graph in Fig. 2.2, Alice needs to issue only one certificate to VeriSign and Bob needs to issue only one certificate to CertPlus. Alice can then use the two certificates $(Alice, VeriSign)$ and $(VeriSign, Amazon)$ to obtain the public key bk_{Amazon}, and so can securely send messages to Amazon. Also, Bob can use the two certificates $(Bob, CertPlus)$ and $(CertPlus, Visa)$ to obtain the public key bk_{Visa}, and then can securely send messages to Visa.

For Alice to use the certificate $(VeriSign, Amazon)$, Alice needs to verify the certificate first through the following four steps.

i. Find the public key of VeriSign: In this example, Alice already has a key that she believes to be the correct public key of VeriSign, in certificate $(Alice, VeriSign)$.

ii. Compute the hash (message digest): Alice computes the hash of all the information included in certificate $(VeriSign, Amazon)$.

iii. Decrypt the included hash with the public key of VeriSign: Alice decrypts

the included hash in certificate $(VeriSign, Amazon)$ with the public key of VeriSign in certificate $(Alice, VeriSign)$.

iv. Compare the two hashes: Alice compares the two hashes from the second and third steps. If these two hashes match, this certificate $(VeriSign, Amazon)$ is successfully verified.

As far as the verification is concerned, the issuer could encrypt the whole certificate with its private key. However, the public key decryption is computationally expensive, so it is easier to compute the hash of a certificate and decrypt just the included hash rather than decrypting the whole certificate. Moreover, since hash functions are one way, users can be assured that using the hash instead of the whole certificate does not compromise the verification.

As discussed above, the certificate issuance is not computationally expensive, but finding the correct public key of a subject may be difficult. The intermediaries in Fig. 2.2 reduce the number of certificates that Alice needs to issue. Instead Alice needs to verify certificates issued by intermediaries (certificate authorities) by computing hashes and decrypting hashes. After verifying the certificates, Alice can learn the public keys of websites and use the keys for communicating securely with the websites.

In general, for users u and v in a certificate graph G, if u wishes to send messages securely to v, then there must be a "chain" from u to v in G. Certificate chains are defined as follows:

A simple path from a source u to a destination v in a certificate graph G is called a *chain* from u to v in G. u is the *source* of the chain and v is the *destination* of the chain. When source u wishes to communicate securely with destination v, source u needs to find a chain from u to v.[1] Once u finds a chain, it needs to verify

[1]There are certificate systems where u needs to find more than a chain from u to v, but we assume the minimum requirement of one chain here. More complicated systems will be discussed in Chapter 4.

all the certificates in the chain to find the public key of destination v. The chain from u to v through $v_0 \cdots v_k$ consists of certificates (u, v_0), (v_0, v_1), $\cdots$, (v_{k-1}, v_k), (v_k, v). The verification of each certificate (v_i, v_{i+1}), $0 \leq i \leq k$ is done as follows:

i. Find the public key of v_i: By the time user u gets to certificate (v_i, v_{i+1}), user u must have verified all the certificates in the chain from u to v_i. From the certificate (v_{i-1}, v_i), user u can obtain the public key of v_i.

ii. Compute the hash (message digest): User u computes the hash of all the information included in certificate (v_i, v_{i+1}).

iii. Decrypt the included hash with bk_{v_i}: User u decrypts the included hash in the certificate with bk_{v_i} found in the first step.

iv. Compare the two hashes: User u compares the two hashes from the second and third steps. If they match, this certificate is successfully verified.

When certificate (v_i, v_{i+1}) is successfully verified, user u moves on to (v_{i+1}, v_{i+2}). Once all the certificates in the chain are verified successfully, source u can obtain the public key of destination v from the last certificate of the chain.

For users u and v in a certificate graph G, if u wishes to securely send messages to v, then there must be a chain from u to v in G. On the other hand, there may be a chain from u to v even though u does not need to securely send messages to v. Fig. 2.3 shows the six chains that are needed to support the secure communications between the two sources and the six destinations in Fig. 2.1. Note that there is a certificate ($VeriSign$, $Amex$) in the certificate graph in Fig. 2.2 that is not needed to support secure communication between any source and any destination in Fig. 2.1. Since Alice does not need to securely communicate with Amex, the certificate chain ($Alice$, $VeriSign$),($VeriSign$, $Amex$) in the certificate graph in Fig. 2.2 is not included in Fig. 2.3.

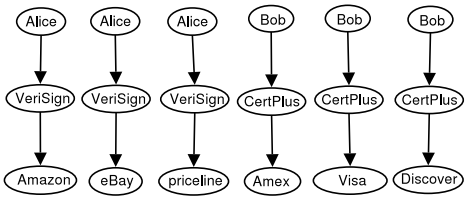


Figure 2.3: Certificate chains from Fig. 2.2

The certificates in each chain need to be "dispersed" between the source and destination of the chain such that if a source u wishes to securely send a message to a destination v then u can obtain the public key of v from the set of certificates stored in u and v. (Note that to "store a certificate in a user" does not necessarily mean that the user has a local copy of the certificate. Rather, it means that the user only needs to know where to find the certificate, if a need for that certificate arises, either in its local storage or in a remote location.)

For example, assume that each source in Fig. 2.3 stores its certificate to the corresponding intermediary, and that each destination in Fig. 2.3 stores the certificate from its corresponding intermediary to itself. Thus,

Alice	stores the certificate ($Alice$, $VeriSign$),
Bob	stores the certificate (Bob, $CertPlus$),
Amazon	stores the certificate ($VeriSign$, $Amazon$),
eBay	stores the certificate ($VeriSign$, $eBay$),
priceline	stores the certificate ($VeriSign$, $priceline$),
Amex	stores the certificate ($CertPlus$, $Amex$),
Visa	stores the certificate ($CertPlus$, $Visa$), and
Discover	stores the certificate ($CertPlus$, $Discover$)

In this case, if Alice wishes to securely send messages to priceline, then Alice can use the two certificates stored in Alice's computer and priceline website to obtain the public key of priceline and securely send the messages to priceline. Certificates

that are not part of any chain are not stored because they are not needed. This is illustrated by the certificate ($VeriSign$, $Amex$), which appears in Fig. 2.2 but is not stored in Amex.

Note that the intermediary, in this case VeriSign, needs to communicate with Alice or priceline for them to securely communicate with each other. In this particular example there is only one intermediary, VeriSign, so it may not be too hard for Alice to contact VeriSign. However, one can imagine that the chain could be arbitrarily longer than 2, and in a such case, it would be rather inefficient if the source of the chain need to contact all the users appearing in the chain. Certificate dispersal, defined in Chapter 3.1 more formally, assigns certificates to users such that source and destination of a chain could find all the certificates in the chain without contacting any other user.

2.3 Certificate Expiration or Revocation

Certificates' lifetime ends when a certificate is either expired or revoked. If the issuer of a certificate had a specific expiration date in mind, then the expiration date becomes part of the certificate. Other users may verify that the certificate has not expired using this expiration date. If the current time is after the expiration date, then other users may choose not to use the certificate nor the public key introduced by the certificate. The issuer can have some control over the usage of a certificate it issued by controlling the expiration date. The later the expiration date is, the longer the certificate may be used.

For a certificate system to be able to control usage with expiration dates, the users' clocks must be synchronized. Imagine a certificate whose expiration date of June 30, 2006. However, if a user has set a wrong time to its system clock, then this user may continue using the certificate even after all other users stop using this certificate. Therefore, for a certificate system to rely on an expiration, some form

of clock synchronization is required.

Certificate revocation is necessary when a certificate becomes invalid before its expiration date comes. There are two reasons of revocation:

i. Incorrect public key of the subject: The issuer intentionally or accidentally signed a certificate with an incorrect public key of the subject.

ii. Revealed private key of the issuer: The private key of the issuer was revealed to an adversary and the certificate may have been issued by the adversary, not by the specified issuer in the certificate.

Certificate revocation in both cases is necessary not only for the issuer but for other users as well. Other users who know the public key of the issuer may learn an incorrect public key of the subject in either case. In the case of revealed private key, the legitimate owner of the private key may revoke the corresponding public key altogether instead of revoking each certificate signed by the revealed private key.

If there is a Certificate Authority (CA) in the system, the Certificate Authority may publish a Certificate Revocation List (CRL). This list contains all the certificates that need to be revoked but have not expired. The list is signed by the CA's private key so that the users in the system may verify the integrity of the list. The delivery of the list may be part of a periodic update sent by CA to all the users in the system. For example, Microsoft Windows updates contains the update on the keys used by Microsoft to sign third party device driver software. The list may also be published in a well-known location, for example the CA's homepage, so that the users may download the list between periodic updates.

If there is no CA in the system, then any issuer in the system may publish its own "revocation certificate". This certificate is signed by the private key of the issuer, and contains either the public key of the issuer or information on a particular certificate. If the public key of the issuer is included, then other users

may stop using the included public key of the issuer, which effectively revoke all the certificates signed by the matching private key of the issuer. (The issuer may get a new pair of public and private key and publish certificates if it wishes.) If the revocation certificate includes information on a particular certificate, it may contain a certificate identification number if it exists, or any unique information of the certificate to be revoked.

This revocation certificate can be dispersed as a regular certificate. If a dispersal is computed periodically by a specific user, then any issuer who issues a revocation certificate send the revocation certificate to the specific user. Then the user can simply ignore the certificate(s) to be revoked according to the revocation certificate. If the revocation certificate contains a public key of an issuer, then all the certificates issued by the corresponding private key will not be dispersed to any user. If the revocation certificate is for a particular certificate, then the revoked certificate will not be dispersed to any user. If the dispersal is not computed by any specific user, then revocation certificates can be dispersed using the dynamic dispersal protocol in Section 3.6, as a regular certificate.

Dispersal of certificate chains and its cost are defined in Chapter 3.1. In Section 3.2, we show that finding an optimal dispersal of any set of chains is NP-complete. Thus it becomes of interest to characterize the special cases of practical interest where the problem can be solved efficiently, as well as effective heuristic algorithms to solve general instances of problems. Subsequently, we identify special classes of chain sets that are of practical interests and devise polynomial-time algorithms that compute optimal dispersals for each class. For instance, the example dispersal above reflects the certificate dispersal in Secure Socket Layer (SSL). Such chain sets are defined as "short" chain sets in Section 3.5, and we present an algorithm that computes an optimal dispersal of any given short chain set.

Chapter 3

Certificate Dispersal

3.1 Certificate Dispersal

In this section, we introduce definitions and notations to describe the optimal dispersal and prove four theorems of the properties of a certificate dispersal.

A *certificate graph* G is a directed graph in which each directed edge, called a *certificate*, is a pair $(u,\ v)$, where u and v are distinct nodes in G. Note that according to this definition no certificate has the same node as both its issuer and subject.

A simple directed path of certificates $(v_0,\ v_1)$, $(v_1,\ v_2)$, $\cdots$, $(v_{k-1},\ v_k)$ in a certificate graph G, where the nodes $v_0,\ v_1,\ \cdots,\ v_k$ are all distinct, is called a certificate *chain* from v_0 to v_k in G.

A *dispersal* D of a certificate graph G assigns a set of certificates in G to each node in G such that the following condition holds. The certificates in each chain from a node u to a node v in G are in the set $D.u \cup D.v$, where $D.u$ and $D.v$ are the two sets of certificates assigned by dispersal D to nodes u and v, respectively.

Let D be a dispersal of a certificate graph G. The *cost* of dispersal D, denoted $cost.D$, is the average number of certificates assigned by dispersal D to each node

in G:

$$cost.D = \frac{1}{n}(\sum_{v \text{ in } G} |D.v|),$$

where n is the number of nodes in G.

A dispersal D of a certificate graph G is *optimal* if and only if for any other dispersal D' of the same certificate graph G, $cost.D \leq cost.D'$.

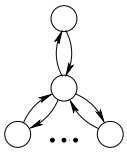


Figure 3.1: A star certificate graph

For example, consider the star certificate graph in Fig. 3.1. This graph can be dispersed as follows. If v is the center node, then $D.v = \{\}$. Otherwise, $D.v = \{(v, \text{center node}), (\text{center node}, v)\}$. The cost of this certificate dispersal is $\frac{2(n-1)}{n}$, where n is the number of nodes in this graph.

Theorem 1 (Upper Bound on Dispersability Cost) *For any certificate dispersal D of a certificate graph G with n nodes,*

$$cost.D \leq n - 1$$

Proof: In Section 3.4, we present a certificate dispersal algorithm F_{full} that assigns to every node v in a certificate graph G, the certificates in a outgoing spanning tree rooted at v. Let D_{full} be the dispersal of G computed by F_{full}. Because each outgoing spanning tree in a certificate graph G has at most $n-1$ certificates, where n is the number of nodes in G, for any node u in G, $|D_{full}.u| \leq n - 1$.

$$cost.D_{full} = \frac{1}{n}(\sum_{v \text{ in } G} |D_{full}.v|) \leq \frac{1}{n}n(n-1) = n - 1$$

For an optimal dispersal D of G,

$$cost.D \leq cost.D_{full} \leq n - 1$$

$$cost.D \leq n - 1$$

■

For strongly-connected graphs and directed graphs, Zheng, Omura, Uchida, and Wada presented algorithms that compute optimal dispersals in [38]. The same authors also showed the tight upper bounds in these two classes of certificate graphs. For a strongly connected graph G, the upper bound is $O(d + e/n)$, where d is the diameter of G, e is the number of edges in G, and n is the number of nodes in G. For a directed graph G', the upper bound on $cost.G'$ is $O(p \times d' + e'/n')$, where p is the number of strongly connected components of G', d' is the maximum diameter of strongly connected components of G', e' is the number of edges in G', and n' is the number of nodes in G'.

A dispersal may be defined on the set of chains that are actually in use, which is a subset of all the chains in a certificate graph. A set of chains in a certificate graph G is called a *chain set* of G. For a chain from node v_0 to another node v_k, node v_0 is called the *source* of the chain and node v_k is called the *destination* of the chain.

A *dispersal* D of a chain set CS assigns a set of certificates in CS to each source node and each destination node in CS such that the following condition holds. The certificates in each chain from a source node u to a destination node v in CS are in the set $D.u \cup D.v$, where $D.u$ and $D.v$ are the two sets of certificates assigned by dispersal D to nodes u and v, respectively. Thus, given a chain in CS, the source node u and the destination node v of the chain can find all the certificates in the chain in the set $D.u \cup D.v$. When the source node u and the destination node

v need to search for a chain from u to v, then they can simply merge $D.u$ and $D.v$ to construct a certificate graph $G_{u,v}$, and search for a simple path from u to v in $G_{u,v}$. If there is a simple path from u to v in $G_{u,v}$, then this path is a certificate chain from u to v. On the other hand, if there is no path from u to v in $G_{u,v}$, then nodes u and v recognize that there was no certificate chain in the given CS.

Dispersal of a chain set is useful for many types of systems. We discuss three example types of systems here.

i. Deployed systems: In a deployed system, all the certificates are dispersed among the nodes in the system before the nodes start on a particular mission. For example, consider mobile units participating in a military operation. Chains that can be used for authentication are carefully chosen and dispersed. Each unit stores the assigned set of certificates by a dispersal of chosen chains. The units are deployed in mission and when a unit needs to authenticate another unit, they do not have guarantee that any other unit will be available. Thanks to dispersal, these two nodes can use the certificates stored in each unit to find a certificate chain from one to the other. Many military applications fit in this type of systems.

ii. Client-Server systems: In a client-server system, there are only a limited number of certificate authorities that issue certificates. In such systems, it is not necessary to collect all the certificates to optimally disperse them. For example, in Secure Socket Layer (SSL) systems, VeriSign is one of the few certificate authorities. A server, for example Amazon.com, does not need to know all the certificates in the system but only stores the certificate $(Amazon.com, VeriSign)$. This is an optimal dispersal (more details are in Section 3.5) of this SSL system.

iii. Evolving systems: In an evolving system where certificates may be issued

and revoked during the execution of the system, the system can start with an optimal dispersal of such system and gradually diverge from the dispersal. Even when the system diverges from its dispersal, it is still beneficial to start with an optimal dispersal as long as the changes in certificates are not a major portion of certificates in the system. Moreover, the dynamic dispersal protocol in [20] disperses newly issued certificates and revocation certificates so that the system stabilizes back to dispersal.

The definitions of the cost of a dispersal of a chain set and its optimality are defined similarly to those in a certificate graph.

Let D be a dispersal of a chain set CS. The *cost* of dispersal D, denoted $cost.D$, is the sum of the number of certificates in the sets assigned by dispersal D to every source or destination node in CS.

$$cost.D = \sum_{v \text{ is a source or destination node in } CS} |D.v|$$

A dispersal D of a chain set CS is *optimal* if and only if for any other dispersal D' of the same chain set CS,

$$cost.D \leq cost.D'$$

In other words, an optimal dispersal D of a chain set CS minimizes the average number of certificates stored in each node.

Let (u, v) be a certificate that appears in one or more chains in a chain set CS, and let D be a dispersal of CS. The *location set* of certificate (u, v) assigned by D, denoted $D(u, v)$, is defined as a set of all nodes x such that (u, v) is in the set of certificates $D.x$. It is straightforward to show that the cost of dispersal D equals $\sum_{(u,v) \in CS} |D(u, v)|$.

The location set $D(u, v)$ of a certificate (u, v) assigned by a dispersal D of a

chain set CS is *optimal* if and only if for any other dispersal D' of CS, $|D(u,v)| \leq |D'(u,v)|$.

Theorem 2 *Let D be a dispersal of a chain set CS. If D is optimal, then for every certificate (u,v) in CS the location set $D(u,v)$ is optimal.*

Proof: The proof is by contradiction. Assume that D is optimal, and there exists another dispersal D' of CS where for some certificate (u,v) in CS, $|D(u,v)| > |D'(u,v)|$.

Now consider the following assignment of certificates to each node in CS.

$$D''(x,y) := \begin{cases} D'(x,y) & \text{if } (x,y) = (u,v), \\ D(x,y) & \text{if } (x,y) \neq (u,v) \end{cases}$$

Note that D'' is a dispersal of CS. This is true because for any chain from a node i to another node j in CS, all the certificates in the chain are in $D''.i \cup D''.j$. Consider a certificate (x,y) in the chain from i to j in CS, where $(x,y) \neq (u,v)$. $D(x,y)$ contains node i or node j by the definition of dispersal, so $D''(x,y)$ contains node i or node j. In other words, any certificate (x,y) in a chain from node i to node j in CS, where $(x,y) \neq (u,v)$, is in $D''.i \cup D''.j$. Similarly, for certificate (u,v), if (u,v) is in a chain from i to j in CS, $D'(u,v)$ contains node i or node j by the definition of dispersal, so $D''(u,v)$ contains node i or node j. In other words, if certificate (u,v) is in a chain from node i to j in CS, then (u,v) is in $D''.i \cup D''.j$. Therefore, for any given chain from a node i to another node j in CS, all the certificates in the chain are in $D''.i \cup D''.j$. Thus, D'' is a dispersal of CS.

The cost of dispersal D'' is computed as follows.

$$cost.D'' = \sum_{v \in CS} |D''.v| = \left(\sum_{(x,y) \in CS, (x,y) \neq (u,v)} |D(x,y)| \right) + |D'(u,v)|$$

By the assumption $|D'(u,v)| < |D(u,v)|$,

$$cost.D'' = \left(\sum_{(x,y)\in CS,(x,y)\neq(u,v)} |D(x,y)| \right) + |D'(u,v)|$$
$$< \left(\sum_{(x,y)\in CS,(x,y)\neq(u,v)} |D(x,y)| \right) + |D(u,v)| = cost.D$$

Thus, the cost of dispersal D'' is less than the cost of dispersal D contradicting the assumption that D is an optimal dispersal. ■

Therefore, the location set $D(u,v)$ assigned by an optimal dispersal D is optimal for every certificate (u,v) in CS.

Theorem 3 *Let D be a dispersal of a chain set CS. If for every certificate (u,v) in CS the location set $D(u,v)$ is optimal, then D is an optimal dispersal of CS.*

Proof: The proof is by contradiction. Let D be a dispersal for a chain set CS and for every certificate (u,v) in CS the location set $D(u,v)$ is optimal. Also, let D' be another dispersal of CS where $cost.D' < cost.D$. By the definition of the cost of dispersal,

$$\sum_{(u,v)\in CS} |D'(u,v)| = cost.D' < cost.D = \sum_{(u,v)\in CS} |D(u,v)|$$

Thus, there must be at least one certificate (u,v) in CS such that $|D'(u,v)| < |D(u,v)|$. This contradicts the definition of an optimal location set of (u,v). ■

Therefore, if $D(u,v)$ is optimal for every certificate (u,v) in a chain set CS, then D is an optimal dispersal of CS.

3.2 NP-Completeness Proof

In this section, we show that the chain dispersal problem is NP-Complete by a reduction from the vertex cover problem. For convenience, these two problems are described below.

- The Vertex Cover (VC) Problem: Given a connected graph G and a positive integer k, we ask if there exists a vertex cover of size $\leq k$. Any instance of this problem can be represented by the pair (G, k). For directed graphs, the VC problem can be defined similarly by ignoring the directions associated with the arcs; the resulting problem on directed graphs remains NP-complete.

- The Certificate Dispersal (CD) Problem: Given a chain set CS and a positive integer m, we ask if there exists a dispersal D of CS such that $cost.D \leq m$. Any instance of this problem can be represented by the pair (CS, m).

Theorem 4 *CD is NP-Complete.*

Proof: First, we show that CD is in NP. Given an instance (CS, m) of CD, and a dispersal D of CS with $cost.D \leq m$, one can verify in polynomial-time that indeed D is a dispersal of CS and $cost.D \leq m$. To verify that D is a dispersal of CS, one checks that all the certificates in each chain from a node u to another node v in CS are in $D.u \cup D.v$. Once D is verified as dispersal, $cost.D$ is computed as the sum of $|D.u|$ for each node u in CS and can be compared to m. The time complexity of this verification step is $O(p \times n)$, where p is the number of chains in the chain set and n is the length of the longest chain in CS.

Second, we show that VC reduces to CD in polynomial-time. Given an instance (G, k) of VC, we construct an instance (CS, m) of CD such that the CD instance has a yes answer if and only if the given VC has a yes answer. The construction is as follows:

i. For each edge (u, v) in G, CS has a chain $(u, x)(x, y)(y, v)$ of length 3.

ii. Let n^+ be the number of nodes that have outgoing edges in G, and n^- be the number of nodes that have incoming edges in G. Set $m = n^+ + n^- + k$.

(CD $\Leftarrow$ VC) We now show that if the instance (G, k) of VC has a yes answer, then the corresponding instance (CS, m) of CD has a yes answer. Let X be a vertex cover of G, where $|X| \leq k$. For each node u in the cover X, assign certificate (x, y) in CS to $D.u$. For each node u in G, if there exists (u, x) in CS, then assign certificate (u, x) to $D.u$. For each node v in G, if there exists (y, v) in CS, then assign certificate (y, v) to $D.v$. In the following two steps, we prove that D is a dispersal of CS whose cost is at most m.

i. D is a dispersal of CS: For any chain in CS from a node u to a node v, the chain consists of three certificates (u, x), (x, y), and (y, v). Certificate (u, x) is stored in $D.u$ and certificate (y, v) is stored in $D.v$. For certificate (x, y), (x, y) is stored in every node in the vertex cover of G. By the definition of the vertex cover, for each edge (u, v) in G, the vertex cover contains node u or node v. Certificate (x, y) is assigned to every node in the vertex cover of G, so (x, y) is stored in $D.u$ or $D.v$. Thus, every certificate in the chain from u to v is stored in $D.u \cup D.v$, as required by the definition of dispersal.

ii. $cost.D \leq m$: For each node u in G that has any outgoing edges, there is certificate (u, x) in CS that is assigned only to node u by D. Similarly, for each node v in G that has any incoming edges, there is certificate (y, v) in CS that is assigned only to node v by D. For certificate (x, y), (x, y) is assigned to all the nodes in the vertex cover, so (x, y) is assigned to at most k nodes. In total, $cost.D$ is at most $m = (k + n^+ + n^-)$.

The above argument shows that D is a dispersal of constructed CS and $cost.D \leq m$. This proves that if an instance of VC (G, k) has a yes answer, then

the corresponding instance of CD (CS, m) has a yes answer.

(CD $\Rightarrow$ VC) We now show that if the constructed instance (CS, m) of CD has a yes answer, then the given instance (G, k) of VC has a yes answer. Let D be a dispersal of CS, where $cost.D \leq m$. For every edge (u, v) in G, there is chain $(u, x)(x, y)(y, v)$ in CS. For certificates (u, x) and (y, v), they will be assigned to at least one node, so $|D(u, x)| \geq 1$ and $|D(y, v)| \geq 1$. The number of such (u, x) certificates is n^+ and the number of such (y, v) certificates is n^-. So certificate (x, y) is assigned to at most k nodes, where k is $m - n^+ - n^-$. In other words, $|D(x, y)| \leq k$.

Now, for each edge (u, v) in G, there is chain $(u, x)(x, y)(y, v)$ in CS, and (x, y) is stored in $D.u \cup D.v$. In other words, for each edge (u, v) in G, the location set of $D(x, y)$ contains node u or node v. Therefore, the location set of $D(x, y)$ is a vertex cover of G. The size of the location set $D(x, y)$ is at most k, so the size of the vertex cover is at most k, and the instance (G, k) of VC has a yes answer.

In conclusion, the above proof shows that CD is in NP and VC reduces to CD in polynomial-time. Therefore, CD is NP-Complete. ■

In the light of the above complexity result, it becomes of importance to identify special classes of chain sets of practical interest for which the problem can be solved efficiently. This direction is pursued in the following cases.

i. *Short chain sets*: In Section 3.5, we start by investigating the class of chain sets, where each chain is of length at most 2. This class of chain sets is the one currently being used in the Secure Socket Layer (SSL) protocol.

ii. *Disconnected chain sets*: In Section 3.5, we investigate the class of chain sets where for a given certificate, no node can be both the source and the destination of any chain that contains this certificate. This reflects a system where the authentication is needed in an asymmetric manner. For example, when there are clients and servers in the system, one can imagine that clients would

use certificates to authenticate servers, while servers would use passwords to authenticate clients. Such asymmetric systems can be represented as this class of chain sets.

iii. *Concise graphs*: In Section 3.4, we investigate the class of chain sets where the chains are derived from acyclic certificate graphs. This class reflects systems where the need for authentication is uni-directional. For example, any hierarchical system where a lower level user is authenticated by a higher level user, but not the other way around, would be represented by an acyclic certificate graph.

For all these three classes of chain sets, we present polynomial-time algorithms that compute optimal dispersals of chain sets in each class and prove their optimality.

Also in Section 3.5, we identify two classes of parameterized chain sets that are defined using an integer parameter k. In the first class, each chain set has at most k chains with 3 or more certificates. In the second class, each chain set has at most k nodes that may act both as sources and destinations. For both classes, we obtain polynomial-time algorithms that compute optimal dispersals when k is fixed.

3.3 Heuristic Dispersal Algorithms

3.3.1 Full Tree Algorithm for Certificate Dispersal

Before we introduce our first certificate dispersal algorithm, we need to introduce the following definition of compact chain sets.

Let G be a certificate graph and v be a node in G. A *compact chain set* for v, denoted $S.v$, is a set of chains in G that satisfies the following three conditions.

i. If G has no chains that starts at v, then $S.v$ is empty.

ii. If G has a chain from v to w, then $S.v$ has exactly one shortest chain from v to w.

iii. If $S.v$ has a chain, then $S.v$ also has every nonempty prefix of this chain.

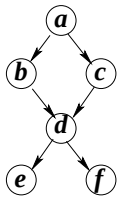


Figure 3.2: The diamond certificate graph

As an example, consider the *diamond* certificate graph in Fig. 3.2. In this graph, there are no certificate chains that start at node e or f, and the compact chain sets for node e and f are both empty:

$$S.e = \{\},\ S.f = \{\}$$

The compact chain set for node d has two chains:

$$S.d = \{< (d, e) >, < (d, f) >\}$$

Also the compact chain set for each of the two nodes b and c has three chains:

$$S.b = \{< (b,d) >, < (b,d);(d,e) >, < (b,d);(d,f) >\}$$

$$S.c = \{< (c,d) >, < (c,d);(d,e) >, < (c,d);(d,f) >\}$$

The compact chain set for node a has five chains:

$$\begin{aligned} S.a = \{ &< (a,b) >, < (a,c) >, < (a,c);(c,d) >, \\ &< (a,c);(c,d);(d,e) >, < (a,c);(c,d);(d,f) >\} \end{aligned}$$

The following two comments are in order. First, each compact chain set $S.v$ for a node v defines a maximal, shortest-path, outgoing tree rooted at node v in the certificate graph. Second, it is possible to have two or more distinct compact chain sets for a node. For example, a second compact chain set for node a in the certificate graph in Figure 4 is as follows:

$$\begin{aligned} \{ &< (a,b) >, < (a,c) >, < (a,b);(b,d) >, \\ &< (a,b);(b,d);(d,e) >, < (a,b);(b,d);(d,f) >\} \end{aligned}$$

Using the above definition of a compact chain set, we are now ready to present our first certificate dispersal algorithm, called the *full tree algorithm* and denoted F_{full}. This algorithm assigns to every node v all the certificates in a compact chain set $S.v$ for v. In other words,

$F_{full}.(G,v)$ = the set of all certificates that exist in a compact chain set $S.v$ for v.

Lemma 1 *F_{full} is a certificate dispersal algorithm.*

Proof: We show that F_{full} satisfies the two conditions of a certificate dispersal

algorithm, connectivity and completeness. First, if there is a chain from u to v in G, then at least one of the shortest chains from u to v is in $S.u$ by condition ii in the definition of compact chain set. Second, any certificate $(u,\ v)$ in G is in $S.u$ since it is the shortest chain from u to v. By the definition of F_{full}, the certificate $(u,\ v)$ is in $F_{full}.(G, u)$. Therefore, F_{full} satisfies two properties of connectivity and completeness. ■

Next, we show that the dispersal algorithm F_{full} is far from being efficient. First, we show in Lemma 5 that the cost of applying F_{full} to any strongly connected certificate graph meets the upper bound on dispersability cost. Second, we show in Lemma 6 that the cost of applying F_{full} to any hourglass certificate graph is within a factor of four from the upper bound on dispersability cost. A certificate graph in Fig. 3.3 is an example hourglass certificate graph. This graph has n nodes and $n-1$ certificates, where n is odd, arranged in an hourglass shape with one center node, $(n-1)/2$ input nodes, and $(n-1)/2$ output nodes.

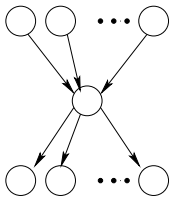


Figure 3.3: An hourglass certificate graph

Lemma 2 *For any strongly connected certificate graph G with n nodes,*

$$c.(F_{full}, G) = n - 1$$

Proof: The certificate dispersal algorithm F_{full} assigns, to every node v in a certificate graph G, the certificates in a maximal outgoing tree rooted at v. If G is

strongly connected, then any maximal outgoing tree is in fact a spanning tree with $(n-1)$ certificates, where n is the number of nodes in G. Therefore, for any node v in G,

$$|F_{full}.(G,v)| = n-1$$
$$c.(F_{full},G) = \frac{1}{n}(\sum_{v \text{ in } G} |F_{full}.(G,v)|) = n-1$$

■

Lemma 3 *For any hourglass certificate graph G with n nodes (see Fig. 3.3),*

$$c.(F_{full},G) = \frac{n^2+2n-3}{4n} \sim \frac{n}{4}$$

Proof: Recall that any hourglass certificate graph G has one center node, $\frac{n-1}{2}$ input nodes, and $\frac{n-1}{2}$ output nodes.

$$|F_{full}.(G,\text{center})| = \frac{n-1}{2}$$

For every input node v,

$$|F_{full}.(G,v)| = \frac{n-1}{2}+1 = \frac{n+1}{2}$$

For every output node v,

$$|F_{full}.(G,v)| = 0$$

Thus,

$$c.(F_{full},G) = \frac{1}{n}(\frac{n-1}{2} + (\frac{n-1}{2})(\frac{n+1}{2}))$$
$$= \frac{n^2+2n-3}{4n} \sim \frac{n}{4}$$

■

3.3.2 Half Tree Algorithm for Certificate Dispersal

Before we introduce our second heuristic dispersal algorithm, we need to introduce the following definition of consistent compact chain sets.

Let $S.u$ and $S.v$ be two compact chain sets for nodes u and v, respectively, in a certificate graph G. $S.u$ and $S.v$ are *consistent* if and only if for every two nodes x and y in G, if $S.u$ has a subchain that starts at x and ends at y and $S.v$ also has a subchain that starts at x and ends at y then these two subchains are identical.

A collection of compact chain sets $\{S.v | v$ is a node in $G\}$ is *consistent* if and only if any two compact chain sets in the collection are consistent.

We are now ready to present our second certificate dispersal algorithm, called the *half tree algorithm* and denoted F_{half}. This algorithm takes as input a consistent collection of compact chain sets $\{S.v | v$ is a node in a certificate graph $G\}$ and computes a set of certificates $F_{half}.(G, v)$ for every node v in G. Algorithm F_{half} is defined in Algorithm 1.

Lemma 4 *F_{half} is a certificate dispersal algorithm.*

Proof: First, if there is a chain between nodes u and v, then at least one of the shortest chains from u to v is stored in $S.u$. All the certificates in the chain from u to v will be stored in u and v by the definition of F_{half}. Second, any certificate (u, v) in G will be stored in $S.u$ since it will be the shortest chain from u to v. By the definition of F_{half}, the certificate (u, v) is stored either in u or in v. Therefore, F_{half} satisfies two properties of certificate dispersal algorithm. ■

Next, we show in Theorem 5 that in the important case of strongly connected certificate graphs, F_{half} is not less efficient than F_{full}, and in some instances, F_{half} is in fact more efficient than F_{full}. Then in Theorem 6, we show that in the important

case of tree certificate graphs, F_{half} is not less efficient than F_{full}, and in some instances, F_{half} is in fact more efficient than F_{full}. In Lemma 5, we show that in the case of the hourglass certificate graphs F_{half} achieves much less dispersal cost than what F_{full} achieves.

Theorem 5 *For any strongly connected certificate graph G,*

$$c.(F_{half}, G) \leq c.(F_{full}, G)$$

For some strongly connected certificate graph G,

$$c.(F_{half}, G) < c.(F_{full}, G)$$

Proof: Let G be any strongly connected certificate graph, and v be any node in G. The certificates in the set $F_{half}.(G, v)$ define a graph G', which is a subgraph of the original graph G. In G', there can be at most one path from any node to node v, and at most one path from node v to any other node. Graph G' satisfies exactly one of the following two conditions.

i. G' has no cycle.

ii. G' has a cycle, but it has at most $n-1$ nodes.

In the first case, the number of certificates in G' is at most $n-1$, since there is no cycle in G'. In the second case, the number of certificates in G' is also at most $n-1$, which is the number of certificates if all the $n-1$ nodes participate in the cycle. Therefore, $|F_{half}.(G, v)| \leq n-1$.

$$c.(F_{half}, G) = \frac{1}{n} \sum_{v \text{ in } G} |F_{half}.(G, v)| \leq \frac{n(n-1)}{n} = n-1$$

Because G is strongly connected, $c.(F_{full}, G) = n - 1$ by Lemma 2. Therefore,

$$c.(F_{half}, G) \leq c.(F_{full}, G)$$

This completes our proof of the first part of the theorem.

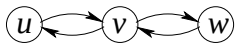


Figure 3.4: The two-ring certificate graph

To prove the second part of the theorem, consider the two-ring certificate graph G'' in Fig. 3.4. This graph is strongly connected and has three nodes. Then by Lemma 2,

$$c.(F_{full}, G'') = n - 1 = 2$$

By applying F_{half} to G'', we get

$$F_{half}.(G'', u) = \{(u, v), (v, u)\}$$
$$F_{half}.(G'', v) = \{\}$$
$$F_{half}.(G'', w) = \{(v, w), (w, v)\}$$

Therefore,

$$c.(F_{half}, G'') = \frac{1}{3}(2 + 0 + 2) = \frac{4}{3} < c.(F_{full}, G'')$$

■

Theorem 6 *For every tree certificate graph T,*

$$c.(F_{half}, T) \leq c.(F_{full}, T)$$

For any complete tree certificate graph G,

$$c.(F_{half}, G) < c.(F_{full}, G)$$

Proof: For any node u in G, the compact chain set $S.u$ of u constructs a maximal shortest-path outgoing tree T_u. Since we may repeatedly store same incoming edges in nodes in F_{half}, $c.(F_{half}, G) \leq \sum_{u \in G} c.(F_{half}, T_u)$, while $c.(F_{full}, G) = \sum_{u \in G} c.(F_{full}, T_u)$. If we can prove $c.(F_{half}, T_u) \leq c.(F_{full}, T_u)$ for any tree T_u, then

$$\begin{aligned} c.(F_{half}, G) &\leq \sum_{u \in G} c.(F_{half}, T_u) \\ &\leq \sum_{u \in G} c.(F_{full}, T_u) = c.(F_{full}, G) \end{aligned}$$

$$c.(F_{half}, G) \leq c.(F_{full}, G)$$

We can prove $c.(F_{half}, T_u) \leq c.(F_{full}, T_u)$ for any tree T_u by induction. When the number of certificates is 2 in the maximal tree, there are 2 possible trees. If the tree looks like Figure 8(a), then $c.(F_{half}, T_u) = c.(F_{full}, T_u) = 2$. If the tree looks like Figure 8(b), then $c.(F_{half}, T_u) = 3$, whereas $c.(F_{full}, T_u) = 2$. Therefore $c.(F_{half}, T_u) \leq c.(F_{full}, T_u)$ holds for any maximal tree T_u with 2 certificates.

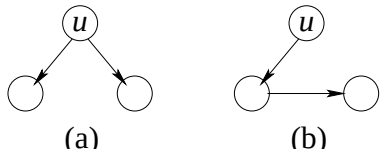
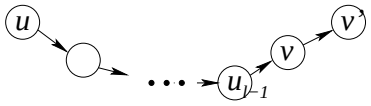


Figure 3.5: Maximal trees with 2 edges

Let's assume that $c.(F_{half}, T_u) \leq c.(F_{full}, T_u)$ holds for trees with up to n certificates. When $n + 1^{th}$ certificate $(v,\ v')$ is added at a node v, then it will increase the chain length from the root node u of the tree to v(This new certificate

has to come with a new subject node v', otherwise it will break the tree property). For a chain from u to a leaf node v' in the given maximal tree T_u, we show that $\sum_{w \in u->v} |F_{half}(T_u, w)| \leq \sum_{w \in u->v} |F_{full}(T_u, w)|$ for any node w on the path from u to v'. The number of certificates stored in the nodes that are not on the path from u to v' will not be affected by this new certificate.



Let l be the chain length from u to v. By the definition of F_{full} algorithm, the increment of $c.(F_{full}, T_u)$ is $l + 1$ because the nodes from u to v will store the new certificate (v, v') locally.

For F_{half}, if a node w is far from v' by even length of chain, for example u_{l-1}, the node w has to store one more outgoing certificates, as the chain length from w to the leaf node v' increases. If $l = 2k$, then the number of such nodes are k. Also $c.(F_{half}, T_u)$ is increased by $F_{half}.(T_u, v')$, which is $k + 1$. Therefore, the increment of $c.(F_{half}, T_u)$ is also $l + 1$, which is equal to that of $c.(F_{full}, T_u)$. If $l = 2k + 1$, then the nodes which stores one more outgoing certificate are k, and $F_{half}.(T_u, v')$ is $k + 1$. But in this case, the certificate from kth node to $k + 1$th node on the chain is not going to be stored as incoming certificate in any nodes any longer. Therefore, $k + 1$ nodes can reduce their $F_{half}.(T_u, v')$ by 1. In total, the increment will be k in $l = 2k + 1$ case.

Since the increment of $c.(F_{half}, T)$ is $l+1$ or $(l-1)/2$ when that of $c.(F_{full}, T)$ is fixed as $l + 1$ when $n + 1$th certificate is added, $c.(F_{half}, T) \leq c.(F_{full}, T)$ holds for any tree T with $n + 1$ number of certificates.

By induction, it is shown that $c.(F_{half}, G) \leq c.(F_{full}, G)$ for any maximal tree T_u for any node u in G. Therefore, $c.(F_{half}, G) \leq c.(F_{full}, G)$ for any tree certificate graph G. This completes our proof of the first part of the lemma.

To prove the second part of the lemma, let h be $\lfloor \log_d n \rfloor$, which is the height

of the tree, where d is the degree of the tree, $d \geq 2$.

$$\begin{aligned} c.(F_{full}, G) &= \sum_{v \text{ in } G} \text{the number of certificates that appear} \\ &\quad \text{in } S.v \\ &= \sum_{1 \leq i \leq h} i * d^i \end{aligned}$$

$$\begin{aligned} c.(F_{half}, G) &= \sum_{v \text{ in } G} \text{the number of certificates that appear} \\ &\quad \text{in } S.v \\ &= \sum_{1 \leq i \leq \lfloor \frac{h}{2} \rfloor} i * d^i + \sum_{\lfloor \frac{h}{2} \rfloor + 1 \leq i \leq h} d^i * (h - i) \\ &\quad + \sum_{\lfloor \frac{h}{2} \rfloor + 1 \leq i \leq h} d^i * (h - i + 1) \\ &= \sum_{1 \leq i \leq \lfloor \frac{h}{2} \rfloor} i * d^i + \sum_{\lfloor \frac{h}{2} \rfloor + 1 \leq i \leq h} d^i * (2h - 2i + 1) \end{aligned}$$

Since

$$\sum_{\lfloor \frac{h}{2} \rfloor + 1 \leq i \leq h} d^i * (2h - 2i + 1) < \sum_{\lfloor \frac{h}{2} \rfloor + 1 \leq i \leq h} i * d^i$$

holds when $d \geq 2$ and $h \geq 1$,

$$c.(F_{half}, G) < c.(F_{full}, G)$$

■

Lemma 5 *For any hourglass certificate graph G with n nodes and e certificates (see*

Figure 3) where n is odd,

$$c.(F_{half}, G) = \frac{e}{n} < c.(F_{full}, G)$$

Proof: Recall that an hourglass certificate graph G with n nodes has one center node, $\frac{n-1}{2}$ input nodes, and $\frac{n-1}{2}$ output nodes. Applying F_{half} to this certificate graph, we get

for every input node u,	$F_{half}.(G, u) = \{(u, c)\}$	
for the center node c,	$F_{half}.(G, c) = \{\,\}$	Therefore,
for every output node w,	$F_{half}.(G, w) = \{(c, w)\}$	

$$c.(F_{half}, G) = \frac{n-1}{n} = \frac{e}{n} < \frac{n}{4} \sim c.(F_{full}, G)$$

■

ALGORITHM 1 : half tree algorithm

INPUT: a certificate graph G
OUTPUT: the half tree dispersal D of G

STEPS:

1: **for** every nonempty $S.v$ in the consistent collection of compact chain sets **do**

2: **let** c denote the longest chain $< (v_0, v_1); \cdots ; (v_{k-1}, v_k) >$ in $S.v$: note that $v_0 = v$;

3: **let** $x := \lfloor \frac{k}{2} \rfloor$;

4: **find** the largest y, $0 \leq y \leq k$, such that all certificates in the prefix $< (v_0, v_1); \cdots ; (v_{y-1}, v_y) >$ are already in $F_{half}(G, v)$;

5: **if** $x \leq y$

6: **then**
store the certificates in every prefix of the subchain $< (v_y, v_{y+1}); \cdots ; (v_{k-1}, v_k) >$ in $F_{half}.(G, w)$ where w is the node at which the prefix ends;

7: **else**

7a: **store** the certificates in the prefix $< (v_y, v_{y+1}); \cdots ; (v_{x-1}, v_x) >$ in $F_{half}(G, v)$;

7b: **store** the certificates in every prefix of the subchain $< (v_x, v_{x+1}); \cdots ; (v_{k-1}, v_k) >$ in $F_{half}.(G, w)$ where w is the node at which the prefix ends;

endif;

8: **remove** chain c from $S.v$;

9: **enddo**;

3.4 Optimal Algorithms for Certificate Graphs

3.4.1 Optimal Dispersal of Reflexive Graphs

In this section we identify a class of certificate graphs called reflexive graphs, and give an algorithm that computes an optimal dispersal of these graphs.

A certificate graph G is called *reflexive* if and only if the following two conditions hold.

i. *Short Cycles* : Every simple directed cycle in G is of length 2.

ii. *Reflexivity* : If there is a certificate from a node u to a node v in G, then G also has a certificate from v to u.

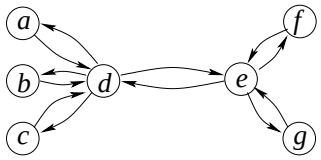


Figure 3.6: An example of a reflexive certificate graph

Fig. 3.6 shows an example of a reflexive graph that has 7 nodes and 12 certificates. Note that there are two opposite direction certificates between the two nodes a and d, and there are no certificates between the two nodes a and b.

A nice feature of reflexive graphs is that there is a certificate chain from any node to any other node in the graph. Thus any node can get the public key of any other node in the graph and can securely send messages to it.

Let G be a reflexive graph. An *undirected version* of G is obtained from G by replacing each pair of opposite direction certificates between two nodes by an undirected edge. For example, an undirected version of the reflexive graph in Fig. 3.6 is shown in Fig. 3.7.

Next we describe an algorithm for optimal dispersal of any reflexive graph G. Note that this algorithm operates on an undirected version G' of G.

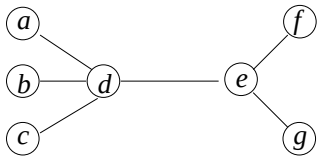


Figure 3.7: An undirected version of the reflexive certificate graph in Fig. 3.6

ALGORITHM 2 : optimal dispersal of a reflexive certificate graph

INPUT: a reflexive certificate graph G
OUTPUT: an optimal dispersal D of G

STEPS:
1: **construct** an undirected version G' of G.
2: **for** each node u in G', $D.u := \{\}$
3: **for** each undirected edge $\{u, v\}$ in G' **do**
4: **compute** the set $R.u$ that contains u and every node x
where there is a simple path between x and u in G'
and this path does not contain the edge $\{u, v\}$
5: **compute** the set $R.v$ that contains v and every node x
where there is a simple path between x and v in G'
and this path does not contain the edge $\{u, v\}$
6: **if** $|R.u| \leq |R.v|$
7: **then** for every node x in $R.u$, $D.x := D.x \cup \{(u, v), (v, u)\}$
8: **else** for every node x in $R.v$, $D.x := D.x \cup \{(u, v), (v, u)\}$

Algorithm 2 can be applied to the reflexive certificate graph in Fig. 3.6 as follows. First, the undirected version of the certificate graph is constructed as shown in Fig. 3.7. For the edge $\{a, d\}$, the two sets $R.a$ and $R.d$ are computed as follows:

$$R.a = \{a\}, R.d = \{b, c, d, e, f, g\}$$

Since $|R.a| = 1 < 6 = |R.d|$, the two certificates $(a,\ d)$ and (d, a) are stored in $D.a$. Similarly, the two certificates $(b,\ d)$ and $(d,\ b)$ are stored in $D.b$ and the two certificates $(c,\ d)$ and $(d,\ c)$ are stored in $D.c$.

For the edge $\{e, f\}$, the two sets $R.e$ and $R.f$ are computed as follows:

$$R.e = \{a, b, c, d, e, g\}, R.f = \{f\}$$

Since $|R.e| = 6 > 1 = |R.f|$, the two certificates $(e,\ f)$ and $(f,\ e)$ are stored in $D.f$. Similarly, the two certificates $(e,\ g)$ and (g, e) are stored in $D.g$.

For the edge $\{d, e\}$, the two sets $R.d$ and $R.e$ are computed as follows:

$$R.d = \{a, b, c, d\}, R.e = \{e, f, g\}$$

Since $|R.d| = 4 > 3 = |R.e|$, the two certificates $(d,\ e)$ and (e, d) are stored in $D.e$, $D.f$, and $D.g$.

The resulting certificate dispersal of the graph is as follows:

$$\begin{aligned}
D.a &= \{(a, d), (d, a)\},\\
D.b &= \{(b, d), (d, b)\},\\
D.c &= \{(c, d), (d, c)\},\\
D.d &= \{\},\\
D.e &= \{(d, e), (e, d)\},\\
D.f &= \{(d, e), (e, d), (e, f), (f, e)\},\\
D.g &= \{(d, e), (e, d), (e, g), (g, e)\}
\end{aligned}$$

The cost of this dispersal is $(2 + 2 + 2 + 0 + 2 + 4 + 4)/7 = 16/7 \sim 2.3$ certificates per node.

Theorem 7 *Given a reflexive certificate graph G, the dispersal D of G computed by Algorithm 2 is optimal.*

Proof: We divide the proof into two parts. First, we show that Algorithm 2

computes a dispersal. Second, we show that D is optimal.

Proof of First Part: By the definition of dispersal in Section 3.1, if all the certificates in each chain from a node u to a node v in G are in set $D.u \cup D.v$, then D is a dispersal of G.

Consider a pair of nodes v_0 and v_k, where there is a certificate chain $(v_0,\ v_1)$, $(v_1,\ v_2)$, $\cdots$, $(v_{k-1},\ v_k)$ from v_0 to v_k in G. For each certificate (v_i, v_{i+1}) in this chain, the two sets $R.v_i$ and $R.v_{i+1}$ are computed by Algorithm 2 for the undirected edge $\{v_i, v_{i+1}\}$. Since there is a chain from v_0 to v_i in G, there is a simple path between v_0 and v_i in G'. Thus, $R.v_i$ contains v_0. Similarly, since there is a simple directed chain from v_{i+1} to v_k in G, there is a simple path between v_{i+1} and v_k in G'. Thus, $R.v_{i+1}$ contains v_k. By steps in line 6-8 in Algorithm 2, (v_i, v_{i+1}) is stored either in all nodes in $R.v_i$ or in all nodes in $R.v_{i+1}$. Because $R.v_i$ contains v_0 and $R.v_{i+1}$ contains v_k, certificate (v_i, v_{i+1}) is stored either in $D.v_0$ or in $D.v_k$. Thus, every certificate (v_i, v_{i+1}) in the chain, is stored in $D.v_0 \cup D.v_k$. Therefore, the chain from v_0 to v_k is stored in the set $D.v_0 \cup D.v_k$. D is a dispersal of G.

For every pair of certificates (u, v) and (v, u) in G, an undirected edge $\{u, v\}$ is constructed in G'. The two certificates (u, v) and (v, u) are stored either in all nodes in $R.u$ or in all nodes in $R.v$, where $R.u$ and $R.v$ are the two sets computed by Algorithm 1 for the undirected edge $\{u, v\}$. By the definition of $R.u$ and $R.v$, $R.u$ contains u and $R.v$ contains v. Thus, by step iii in Algorithm 2, the two certificates (u, v) and (v, u) are either stored in $D.u$ or in $D.v$. Therefore, for every certificate in G, there is a node x in G such that this certificate is in $D.x$. The completeness condition holds.

Proof of Second Part:

Let D' be any other dispersal of a reflexive certificate graph G and let (u, v) be any directed certificate in G. The certificate (u, v) is on every directed chain from a node in $R.u$ to a node in $R.v$, where $R.u$ and $R.v$ are the two sets computed by

Algorithm 2 for the undirected edge $\{u, v\}$. Therefore, D' needs to assign certificate (u, v) to every node in $R.u$ or to every node in $R.v$. In either case, D' yields a dispersal cost that is no less than the dispersal cost of D computed by Algorithm 2. ■

The complexity of Algorithm 2 is $O(en)$, where e is the number of edges in the undirected version of the input reflexive graph and n is the number of nodes in the reflexive graph. Since $e = n - 1$, the complexity of this algorithm is $O(n^2)$.

Note that the star certificate graph in Fig. 3.1 in Section 3.1 is reflexive and so Algorithm 2 can be used to compute an optimal dispersal of this graph. Using Algorithm 2, we obtain the following certificate dispersal for this graph:

$D.v = \{\}$ if v is the center node
$D.v = \{(v,$ center node$),($center node$, v)\}$ otherwise

The cost of this certificate dispersal $= (0 + 2(n - 1))/n$. From Theorem 7, we conclude that this cost is the smallest possible cost of certificate dispersal for the star certificate graph.

3.4.2 Optimal Dispersal of Biased Graphs

In this section, we present an algorithm that computes an optimal dispersal for another class of certificate graphs, called biased graphs. As discussed below, the class of biased graphs is for all practical purposes mutually exclusive from the class of reflexive graphs discussed in the previous section.

A certificate graph G is called *biased* if and only if it satisfies the following two conditions.

i. *Acyclicity* : G has no directed cycles.

ii. *Nonredundancy* : G has at most one certificate chain from any node to any

other node.

From the definitions of reflexive and biased graphs, it follows that every reflexive graph that has one or more certificates is not biased and every biased graph that has one or more certificates is not reflexive. Biased certificate graphs represent many useful certificate systems. For example, a hierarchical certificate system would typically generate a tree-shaped certificate graph. Any directed tree-shaped certificate graph is a biased certificate graph.

Note that a reflexive graph supports secure two-way communication between every two nodes in the graph, whereas a biased graph supports secure one-way communication between some two nodes in the graph. For example, consider the biased graph in Fig. 3.8. This graph supports secure one-way communication from node a to node b and from node a to node c, but it does not support any secure communication between the two nodes b and c.

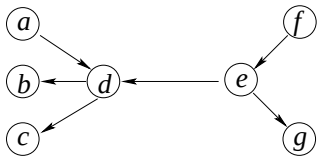


Figure 3.8: A biased certificate graph

Next, we present an algorithm which computes optimal dispersals for the class of biased graphs.

As an example, let us consider the application of the steps in lines 5–7 in Algorithm 3 on the certificate (a, d) in the biased graph in Fig. 3.8. In this case, the two sets $R.a$ and $R.d$ are computed as follows:

$$R.a = \{a\}, R.d = \{d, b, c\}$$

Thus, $|R.a| = 1 < 3 = |R.d|$ and so certificate (a, d) is added only to $D.a$.

ALGORITHM 3 : optimal algorithm of a biased certificate graph

INPUT: a biased certificate graph G
OUTPUT: an optimal dispersal D of G

STEPS:
1: **for** each node u in G, $D.u := \{\}$
2: **for** each certificate (u, v) in G **do**
3: **compute** the set $R.u$ that contains u and every node x where there is a chain from x to u in G
4: **compute** the set $R.v$ that contains v and every node x where there is a chain from v to x in G
5: **if** $|R.u| \leq |R.v|$
6: **then** for every node x in $R.u$, $D.x := D.x \cup \{(u, v)\}$
7: **else** for every node x in $R.v$, $D.x := D.x \cup \{(u, v)\}$

As a second example, consider the application of the steps in lines 5–7 in Algorithm 3 on the certificate (e, g) in the biased graph in Fig. 3.8. In this case, the two sets $R.e$ and $R.g$ are computed as follows:

$$R.e = \{f, e\}, R.g = \{g\}$$

Thus, $|R.e| = 2 > 1 = |R.g|$ and so certificate (e, g) is added only to $D.g$.

Theorem 8 *Given a biased certificate graph G, the dispersal D of G computed by Algorithm 3 is optimal.*

Proof: The proof is similar to that of Theorem 7. ■

3.4.3 Optimal Dispersal of Concise Graphs

In this section, we present an algorithm that computes optimal dispersal for chain sets "derivable" from a class of certificate graphs called concise certificate graphs. A certificate graph G is called *concise* if and only if it satisfies the following two

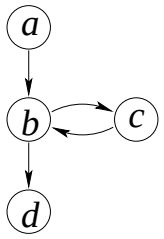
conditions.

i. *Short Cycles* : Every simple directed cycle in G is of length 2.

ii. *Non-redundancy* : G has at most one chain from any node to any other node.

Concise certificate graphs represent many useful certificate systems. For example, a hierarchical certificate system would typically generate a tree-shaped certificate graph. Any tree-shaped certificate graph is a concise certificate graph.

Fig. 3.9(a) shows an example of a concise certificate graph. Note that in a concise graph there can be two opposite direction certificates between two adjacent nodes. We refer to any such pair of certificates as *twins*, and we refer to each one of those certificates as the *twin certificate* of the other. In the concise graph in Fig. 3.9(a), the two certificates (b, c) and (c, b) are twins.



(a)

$\{ \quad (a,b),\ (b,c),\ (c,b),\ (b,d),$
$(a,b)(b,c),$
$(a,b)(b,d),$
$(c,b)(b,d)\ \}$

(b)

Figure 3.9: An Example of Concise Certificate Graph and Derivable Chain Set

A chain set is *derivable* from some certificate graph G if and only if the chain set consists of all the certificate chains in G. For example, the chain set in Fig. 3.9(b) is derivable from the certificate graph in Fig. 3.9(a).

Algorithm 4 computes an optimal dispersal of a concise certificate graph. Consider certificate (b, c) in the example concise certificate graph in Fig. 3.9(a). Algorithm 4 computes the set of nodes from which there is a chain to b, denoted $R.b$, as $\{a, b\}$. Also, Algorithm 4 computes the set of nodes to which there is a chain

ALGORITHM 4 : optimal dispersal of concise certificate graphs

INPUT: a concise certificate graph G
OUTPUT: a dispersal D of the chain set CS derivable from G

STEPS:
1: **for** each node u in G, $D.u := \{\}$
2: **for** each certificate (u, v) in G **do**
3: **compute** the set $R.u$ that contains u and every node x from which there is a chain to u in G and this chain does not contain the twin certificate (v, u)
4: **compute** the set $R.v$ that contains v and every node x to which there is a chain from v in G and this chain does not contain the twin certificate (v, u)
5: **if** $|R.u| \leq |R.v|$
6: **then** for every node x in $R.u$, add (u, v) to $D.x$
7: **else** for every node y in $R.v$, add (u, v) to $D.y$

from c, denoted $R.c$ as $\{c\}$. $|R.b| > |R.c|$, so (b, c) is stored in c. After considering all the certificates in the graph, the example concise certificate graph is optimally dispersed by Algorithm 4 as follows:

$$\{ \quad D.a = \{(a, b)\},\ D.b = \{(c, b)\},$$
$$D.c = \{(b, c)\},\ D.d = \{(b, d)\} \quad \}$$

Theorem 9 *Given a concise certificate graph G, the dispersal D of the chain set CS derivable from G computed by Algorithm 4 is optimal.*

Proof: We divide the proof into two parts. First, we show that Algorithm 4 computes a dispersal D. Second, we show that D is optimal.

Proof of First Part:

We show that the certificate subsets $D.x$, computed by Algorithm 4 for every node x in G, satisfy the condition of dispersal in Section 2.

Consider a pair of nodes v_0 and v_k, where there is a chain (v_0, v_1), (v_1, v_2), $\cdots$, (v_{k-1}, v_k) from v_0 to v_k in G. By the definition of the derivable chain set, the chain from v_0 to v_k is in CS. For each certificate (v_i, v_{i+1}) in this chain, the two sets $R.v_i$ and $R.v_{i+1}$ are computed by Algorithm 4. Since there is a chain from v_0 to v_i in G, $R.v_i$ contains v_0. Similarly, since there is a simple directed chain from v_{i+1} to v_k in G, $R.v_{i+1}$ contains v_k. By line 5-7 in Algorithm 4, (v_i, v_{i+1}) is stored either in all nodes in $R.v_i$ or in all nodes in $R.v_{i+1}$. Because $R.v_i$ contains v_0 and $R.v_{i+1}$ contains v_k, certificate (v_i, v_{i+1}) is stored either in $D.v_0$ or in $D.v_k$. Thus, every certificate (v_i, v_{i+1}) in the chain from v_0 to v_k is stored in $D.v_0 \cup D.v_k$. Hence, D is a dispersal of the chain set CS derivable from G.

Proof of Second Part: The proof is by contradiction. Let D' be another dispersal of CS where $cost.D' < cost.D$. Then there must be such a certificate (u, v) that $|D'(u, v)| < |D(u, v)|$. By the definition of dispersal, (u, v) needs to be stored in $D'.x \cup D'.y$ for every chain from x to y that contains (u, v). By the definition of derivable chain set, certificate (u, v) is used in every directed chain from any node x in $R.u$ to any node y in $R.v$, where $R.u$ and $R.v$ are the two sets computed by Algorithm 4 for certificate (u, v). In other words, $|D'(u, v)| \geq min(|R.u|, |R.v|)$. Since $|D(u, v)| = min(|R.u|, |R.v|)$, $|D'(u, v)| \geq |D(u, v)|$. This contradicts the assumption of $|D'(u, v)| < |D(u, v)|$.

Therefore, D computed by Algorithm 4 is optimal. ■

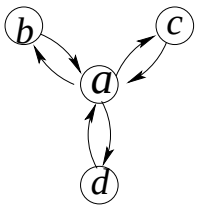
The complexity of Algorithm 4 is $O(en)$, where e is the number of certificates in the input concise certificate graph and n is the number of nodes in the concise certificate graph.

3.5 Optimal Algorithms for Chain Sets

3.5.1 Optimal Dispersal of Short Chain Sets

In the previous section, we proved that computing an optimal dispersal of any chain set, which includes chains whose length is 3 or more, is NP-complete. In this section, we show that there is a polynomial-time algorithm that computes an optimal dispersal of any chain set whose chains are all of length at most 2. This class of chain sets is currently in use in the Internet in Secure Socket Layer (SSL).

A chain set CS is *short* if and only if the length of the longest chain in CS is at most 2. For example, consider the star certificate graph in Fig. 3.10(a). In this certificate graph, assume that each satellite node, b, c, or d, wishes to securely communicate with every other satellite node. Fig. 3.10(b) shows the resulting short chain set.



(a)

$$\{ \quad (b,a)(a,c), \quad (d,a)(a,b), \quad (c,a)(a,b), \quad (c,a)(a,d), \quad (b,a)(a,d), \quad (d,a)(a,c)\}$$

(b)

Figure 3.10: An Example of Short Chain Set

Algorithm 5 computes an optimal dispersal of a short chain set. Consider the certificate (b,a) in the example short chain set in Fig. 3.10. Chains that have (b,a) are $(b,a)(a,c)$ and $(b,a)(a,d)$. So b is the source of every chain that has (b,a). Therefore, Algorithm 5 assigns (b,a) to $D.b$. After considering all the certificates in the short chain set, the optimal dispersal computed by Algorithm 5 as follows:

ALGORITHM 5 : optimal dispersal of short chain sets

INPUT: a short chain set CS
OUTPUT: a dispersal D of CS

STEPS:
1: **for** each node u in CS, $D.u := \{\}$
2: **for** each certificate (u, v) in CS **do**
3: **if** there is a node x such that
the source or destination of every chain that has (u, v) is x
4: **then** add (u, v) to $D.x$
5: **else** add (u, v) to both $D.u$ and $D.v$

$$\{D.a = \{\},\ D.b = \{(a, b), (b, a)\},$$
$$D.c = \{(a, c), (c, a)\},\ D.d = \{(a, d), (d, a)\}\}$$

Theorem 10 *Given a short chain set CS, the dispersal D of CS computed by Algorithm 5 is optimal.*

Proof: The proof consists of two parts. First, we show that Algorithm 5 computes a dispersal D. Second, we show that D is optimal.

Proof of First Part:

By the definition of dispersal in Section 2, if all the certificates in each chain from a source node u to a destination node v in CS are in set $D.u \cup D.v$, then D is a dispersal of CS. In other words, if a certificate (u, v) is stored in the source or destination nodes of every chain that contains (u, v), then D is a dispersal.

By Algorithm 5, every certificate (u, v) is stored either in $D.x$ of some node x, or both $D.u$ and $D.v$. Since the maximum length of a chain in CS is 2, every chain that contains (u, v) starts at u or ends at v. Hence if (u, v) is stored in both $D.u$ and $D.v$ then certificate (u, v) is stored in the source or destination node of

every chain that contains (u, v). If (u, v) is stored in node x, then by Algorithm 5 x is either the source node or the destination node of every chain that contains (u, v). Therefore, (u, v) is stored in the source or the destination node of every chain that contains (u, v).

Proof of Second Part:

The proof is by contradiction. Let D be the dispersal of a short chain set CS computed by Algorithm 5 and D' be another dispersal of CS. Assume that $cost.D' < cost.D$. There must be at least one certificate (u, v) such that $|D'(u, v)| < |D(u, v)|$.

Let (u, v) be such a certificate, $|D'(u, v)| < |D(u, v)|$. By Algorithm 5, $|D(u, v)|$ is either 1 (if there exists some node x that is the source or destination node of every chain that has (u, v)) or 2 (otherwise). Therefore, $|D'(u, v)| = 1$ and $|D(u, v)| = 2$, and there exists no node x in CS that is the source or destination node of every chain that has (u, v). By the definition of dispersal, the node w in $D'(u, v)$ should be the source or a destination of every chain that contains (u, v) in CS. This contradicts that there exists no node x in CS such that x is the source or destination node of every chain that has (u, v).

Therefore, $cost.D \leq cost.D'$ for any dispersal D' of CS. Algorithm 5 computes an optimal dispersal of a short chain set CS. ■

The time complexity of Algorithm 5 is $O(ep)$, where e is the number of certificates in the input short chain set and p is the number of chains in the chain set.

3.5.2 Optimal Dispersal of Disconnected Chain Sets

In this section, we identify a special class of chain sets and present an algorithm that computes an optimal dispersal for this class of chain sets in polynomial-time. A chain set CS is *disconnected* if and only if for every certificate (u, v) in CS, the set of source nodes of the chains that contain (u, v) and the set of destination

nodes of the chains that contain (u, v) are disjoint. This reflects a system where the authentication is performed in an asymmetric manner. For example, when there are clients and servers in the system, one can imagine that clients would use certificates to authenticate servers, while servers would use passwords to authenticate clients. Such asymmetric systems can be represented as disconnected chain sets. Fig. 3.11 shows an example of a disconnected chain set.

$$\begin{array}{ll} \{ & (d,a), \\ & (a,b)(b,c), \\ & (a,c)(c,d), \\ & (a,b)(b,c)(c,d)(d,e)\} \end{array}$$

Figure 3.11: An Example of Disconnected Chain Set

(d, a) has the set of source nodes $\{d\}$ and the set of destination nodes $\{e\}$, which are disjoint. (a, b) has the set of source nodes $\{a\}$ and the set of destination nodes $\{c, e\}$, which are disjoint. Every certificate in this chain set has disjoint sets of source and destination nodes.

ALGORITHM 6 : optimal dispersal of disconnected chain sets

INPUT: a disconnected chain set CS
OUTPUT: a dispersal D of CS

STEPS:
1: **for** each node u in G, $D.u := \{\}$
2: **for** each certificate (u, v) in G **do**
3: G'=(V', E') where $V' = \{\}$ and $E' = \{\}$
4: **for** each chain from node x to node y that contains (u, v) **do**
5: add nodes x and y to V'
6: add (x, y) to E'
7: **compute** a minimal vertex cover of the bipartite graph G'
8: **add** (u, v) to each node in the vertex cover

Algorithm 6 computes an optimal dispersal of a disconnected chain set. Consider certificate (a, b) in the example disconnected chain set in Fig. 3.11. Algo-

rithm 6 constructs a bipartite graph G' for certificate (a, b), where $G' = (V', E')$, V'=$\{a, c, e\}$, and E'=$\{(a, c), (a, e)\}$. The vertex cover of minimum size of G' is $\{a\}$. Thus, (a, b) is stored in $D.a$. After considering all certificates in the chain set, the example disconnected chain set is optimally dispersed by Algorithm 6 as follows:

$$\begin{aligned} \{D.a = &\ \{(a,b),(b,c),(c,d)\},\ D.b = \{\},\ D.c = \{\}, \\ D.d = &\ \{(a,c),(d,a)\},\ D.e = \{(d,e)\}\} \end{aligned}$$

Theorem 11 *Given a disconnected chain set CS, the dispersal D of CS computed by Algorithm 6 is optimal.*

Proof: The proof consists of two parts. First, we show that Algorithm 6 produces a dispersal. Second, we show that the resulting dispersal is optimal.

Proof of First Part:

Let $D.u$ be the set of certificates assigned to a node u in CS by Algorithm 6. Consider any certificate (u, v) in a chain from a source node x to a destination node y in CS. By Algorithm 6, since there is a chain from x to y that goes through (u, v), there is an edge (x, y) in G' for (u, v). By the definition of vertex cover, for edge (x, y) in G', node x or node y is in the vertex cover. Therefore, for the chain from x to y, (u, v) is stored in $D.x$ or $D.y$. This is true for all the certificates in the chain from x to y, for any chain in CS. Hence, D satisfies the dispersal condition in Section 2, so D is a dispersal of CS.

Proof of Second Part:

By Theorem 3, if we can find a dispersal D where $D(u, v)$ of every certificate (u, v) in CS is optimal, then D is an optimal dispersal of CS. So we only need to prove that a dispersal computed by Algorithm 6 produces an optimal location set of each certificate in CS. The proof is by contradiction. Assume there is another dispersal D' of CS, where $cost.D' < cost.D$. There must be at least one certificate

(u, v) where $|D'(u, v)| < |D(u, v)|$. For every chain from a node x to a node y that contains (u, v), $D'(u, v)$ should contain x or y. Therefore, $D'(u, v)$ is a vertex cover of the bipartite graph G' constructed for (u, v), where $|D'(u, v)| < |D(u, v)|$. This contradicts that $D(u, v)$ is the vertex cover of minimum size of G' by line 7 in Algorithm 6. Therefore, $D(u, v)$ is an optimal location set of (u, v) for every certificate (u, v) in CS. By Theorem 3, D is optimal. ■

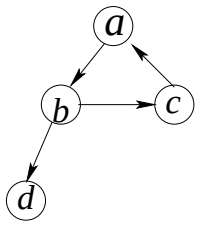
For each certificate (u, v), the graph G' constructed for (u, v) is a bipartite graph. It is because the set of source nodes of the chains that contain (u, v) and the set of the destination nodes of the chains that contain (u, v) are disjoint by the definition of disconnected chain set. Finding a vertex cover in a bipartite graph is a well known problem in graph theory, which takes $O(n'e')$ steps where n' is the number on nodes in G' and e' is the number of edges in G'. In the worst case $n' = n$ and $e' = p$, where n is the number of nodes in CS, and p is the number of chains in CS. Therefore, the time complexity of Algorithm 6 is $O(e \times np)$=$O(enp)$, where e is the number of certificates in CS.

3.5.3 Optimal Dispersal of k-long Chain Sets

In Section 3.2, we showed that computing an optimal dispersal of any chain set, which includes chains of length 3 or more, is NP-complete. If all the chains in a chain set are of length at most 2, i.e. if the chain set is short, then we can use Algorithm 5 in Section 3.5.1 to compute an optimal dispersal of the short chain set. In this section, we consider a more general class of chain sets where there are a fixed number k, $k \geq 1$, of chains of length greater than 2. Consideration of such chain sets is motivated, for instance, by the following example. Consider a hierarchical network made of a number of autonomous systems. Certificate chains within any single autonomous system are expected to be short, whereas certificate chains that span multiple autonomous systems are expected to be long. The chain set of these

autonomous systems contain mostly short *intra*-chains, but may contain a fixed number of long *inter*-chains. Our main result here is a polynomial-time algorithm that computes an optimal dispersal for such chain set for fixed k.

In this section, we present Algorithm 7 that computes an optimal dispersal of a chain set where there are k chains of length greater than 2 for some constant k. We call such sets *k-long* chain sets. Roughly speaking, our general strategy is to consider all possible ways of assigning certificates that appear in long chains to the relevant source and destination nodes, and then handling the remaining short chains with the aid of Algorithm 5. To develop some initial intuition, first we show how to compute an optimal dispersal of an example 1-long chain set in Fig. 3.12(b), and then we show how to generalize for k-long chain sets.



(a)

$\{ \quad (a,b)(b,c),$
$(b,c)(c,a),$
$(c,a)(a,b),$
$(c,a)(a,b)(b,d) \ \}$

(b)

Figure 3.12: An Example of 1-Long Chain Set

Let CS be the 1-long chain set in Fig. 3.12(b), which is a chain set of the certificate graph in Fig. 3.12(a). There is one long chain $(c,a)(a,b)(b,d)$ and three other short chains. There are three types of certificates in this chain set.

i. Certificates used only in long chains: for example, (b,d).

   A certificate of this type can be dispersed either to the source or to the destination of each long chain that contains this certificate. For example, certificate (b,d) in CS is used only in the long chain and needs to be dispersed either to c or to d. This certificate is not used in any other chains, so it does not change

the cost of dispersal whether it is dispersed to c or d.

ii. Certificates used only in short chains: for example, (b, c).

For certificates of the second type, we can use Algorithm 5 in Section 3.5.1 to disperse such certificates. For example, certificate (b, c) is dispersed to node a by Algorithm 5.

iii. Certificates used in both long and short chains: for example, (a, b), (c, a).

Dispersing a certificate of the third type needs to consider every possible assignment of this certificate among sources and destinations of long chains. For example, certificate (a, b) is used in three chains, $(a, b)(b, c)$, $(c, a)(a, b)$ and $(c, a)(a, b)(b, d)$. If we choose to disperse (a, b) to the source c of long chain, then we do not need to disperse (a, b) to any other node in CS, since c happens to be source or destination of all the short chains that contain (a, b). By contrast, if we choose to disperse (a, b) to the destination d of long chain, then we need to disperse (a, b) to other nodes than d since d is neither source nor destination of two short chains $(a, b)(b, c)$ and $(c, a)(a, b)$. In other words, $D(a, b)$ could be either $\{c\}$ or $\{a, b, d\}$, depending on whether (a, b) is assigned to the source or the destination of the long chain. This shows that for each certificate of the third type that is used in both long and short chains, in each assignment of this certificate in sources and destinations of long chains, we need to check which short chains still needs dispersal of this certificate.

After considering all three types of certificates in CS, the resulting optimal dispersal of CS in Fig. 3.12(b) becomes as follows:

$$\begin{aligned}\{ \quad & D.a = \{(b, c)\},\ D.b = \{(c, a)\}, \\ & D.c = \{(c, a), (a, b)\},\ D.d = \{(b, d)\} \quad \}\end{aligned}$$

To extend this solution for 1-long chain set to k-long chain set, we need to define a terminal set of a chain set. A *terminal* set of a chain set CS is a subset of nodes in CS that consists of the source or destination of each chain in CS. For example, the four nodes a, b, c, c are the sources of all four chains in the chain set in Fig. 3.12(b), so $\{a, b, c\}$ is a terminal set of this chain set. Algorithm 7 computes an optimal dispersal of k-long chain sets using this terminal set.

ALGORITHM 7 : optimal dispersal of k-long chain sets

INPUT: a k-long chain set CS
OUTPUT: a dispersal D of CS

STEPS:
1: **for** each node u in CS, $D.u := \{\}$
2: **for** each certificate (u, v) in CS **do**
3: **compute** the chain set LS of all long chains that contain (u, v) in CS
4: **for** each possible terminal set X of LS
5: **for** each node w in CS,
6: **if** $w \in X$ **then** $D_X.w := \{(u, v)\}$ **else** $D_X.w := \{\}$
7: **compute** the chain set S of all the chains that contain (u, v) and their sources and destinations are not in X
8: **run** Algorithm 5 on S and add the resulting location set of (u, v) to D_X
9: **find** D_X with the minimal cost
10: **for** each node u in CS, add $D_X.u$ to $D.u$

Consider (c, a) in the example chain set in Fig. 3.12(b). The set of all long chains that contain (c, a), denoted LS in Algorithm 7, is $\{(c, a)(a, b)(b, d)\}$. For a terminal set $\{c\}$, (c, a) is dispersed to node c and the set of remaining short chains, denoted S in Algorithm 7, becomes $\{(b, c)(c, a)\}$. There is node b that is the source of every chain in S, so (c, a) is dispersed to node b. The resulting dispersal of (c, a), $\{b, c\}$, is an optimal location set of (c, a). After considering every certificate, the dispersal of the example chain set in Fig. 3.12(b) computed by Algorithm 7 becomes the same with the dispersal above, and this dispersal is optimal. Theorem 12 shows

that Algorithm 7 computes an optimal dispersal of a given k-long chain sets.

Theorem 12 *Given a k-long chain set CS, the dispersal D of the chain set CS computed by Algorithm 7 is optimal.*

Proof: We divide the proof into two parts. First, we show that Algorithm 7 computes a dispersal D. Second, we show that D is optimal.

Proof of First Part:

We show that the certificate subsets $D.u$, computed by Algorithm 7 for every node u in CS, satisfy the condition of dispersal in Section 2.

Consider a certificate (u, v). Algorithm 7 computes the chain set LS of for all the long chains that contain (u, v). Algorithm 7 stores (u, v) in every node in a terminal set of LS. By the definition of a terminal set, (u, v) is stored in either source or destination of each long chain in LS. For all the remaining short chains that contain (u, v) in CS, by line 7-9 in Algorithm 7 (same as line 3-5 in Algorithm 5), (u, v) is stored either in $D.w$ for some node w or in $D.u$ and $D.v$. (The rest of proof is same with the optimality proof of Algorithm 5.) For each remaining short chain, the chain that contains (u, v) starts at u or ends at v. Hence if (u, v) is stored in both $D.u$ and $D.v$ then certificate (u, v) is stored in the source or destination node of every remaining chain that contains (u, v). If (u, v) is stored in node w, then by Algorithm 7, then w is either the source node or the destination node of every remaining chain. Therefore, (u, v) is stored in the source or the destination node of every chain that contains (u, v). This is true for any certificate (u, v) in CS. Hence, D is a dispersal of the chain set CS.

Proof of Second Part:

The proof is by contradiction. Let D be the dispersal of a k-long chain set CS computed by Algorithm 7 and D' be another dispersal of CS. Assume that $cost.D' < cost.D$. There must be at least one certificate (u, v) such that $|D'(u, v)| < |D(u, v)|$.

There are three cases of (u, v):

i. (u, v) is a certificate used only in long chains.

ii. (u, v) is a certificate used only in short chains.

iii. (u, v) is a certificate used in both long and short chains.

For case i), Algorithm 7 considers every possible terminal set X of the long chains that contain (u, v). Therefore, the resulting $|D(u, v)| = \min_X |D_X(u, v)|$. By the definition of the terminal set, $D'(u, v)$ has to be a terminal set of the long chains that contain (u, v). In other words, $|D'(u, v)| \geq \min_X |D_X(u, v)| = |D(u, v)|$. Therefore, $|D(u, v)| \leq |D'(u, v)|$

For case ii), Algorithm 7 computes an optimal dispersal of the short chains containing (u, v). The proof is same as the optimality proof of Algorithm 5 for short chain sets. Therefore, $|D(u, v)| \leq |D'(u, v)|$.

For case iii), find a terminal set X of the long chains that contain (u, v), such that $X \subset D'(u, v)$. Since Algorithm 7 considers every possible terminal set of the long chains that contain (u, v), it also computes $D_X(u, v)$ for the found terminal set X, where $X \subset D_X(u, v)$. For the remaining short chains in S, since the sources and destinations of the short chains in S are not in X, so $D'(u, v) \setminus X$ should contain source or destination of each chain in S. Also, Algorithm 5 computes an optimal location set of (u, v) in S. Therefore, $|D_X(u, v) \setminus X| \leq |D'(u, v) \setminus X|$. Since $X \subset D'(u, v)$ and $X \subset D_X(u, v)$, $|D_X(u, v)| \leq |D'(u, v)|$. $|D(u, v)| = \min_X |D_X(u, v)|$, so $|D(u, v)| \leq |D'(u, v)|$.

In all three cases, $|D(u, v)| \leq |D'(u, v)|$, which contradicts the assumption of $|D(u, v)| > |D'(u, v)|$. Therefore, dispersal D computed by Algorithm 7 is optimal.

■

The time complexity of this algorithm is $O(2^k \times ep)$, where k is the number of long chains in CS, e is the number of certificates in CS, and p is the number of

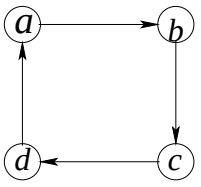
chains in CS. This complexity is computed as follows: the number of terminal sets for k long chains is $O(2^k)$, and for each terminal set, the number of short chains to consider is $O(p)$. We repeat this procedure for e certificates. Since k is a constant, the time complexity becomes $O(ep)$.

3.5.4 Optimal Dispersal of k-Connected Chain Sets

In Section 3.5.2, we presented Algorithm 6 that computes an optimal dispersal of a disconnected chain set. In this section, we investigate more general class of chain sets where there are at most k nodes in the intersection of the source set and the destination set of each certificate in a chain set. We call such chain sets *k-connected* chain sets. This class of chain sets models a client-server system that uses two different authentication methods. As discussed in Section 3.5.2, in some client-server systems, clients authenticate servers via certificates, whereas servers authenticate clients via other means, e.g. passwords. However, there may be a few mutual authentications via certificates between servers. These certificates used by servers may have non-empty intersection of the source and destination sets. Such client-server systems can be represented as k-connected chain sets.

Fig. 3.13(b) shows an example of 1-connected chain set, which is a chain set of the certificate graph in Fig. 3.13(a). For certificate (a, b), the sources of the chains that contain (a, b) are $\{a, c\}$ and the destinations of such chains are $\{b, c, d\}$. The intersection of two sets is $\{c\}$. Similarly, the cardinality of the intersection set is at most 1 for every certificate in this chain set, so the chain set in Fig. 3.13(b) is 1-connected.

Assume that (a, b) is stored in $D.c$ in some dispersal D of this chain set. The remaining chain to be dispersed is $(a, b)(b, c)(c, d)$. Certificate (a, b) can be stored either in $D.a$ or in $D.d$, either of which makes no difference in the dispersal cost. Or, assume that (a, b) is not stored in $D'.c$ in some dispersal D' of this chain set.



(a)

$\{ \quad (a,b)(b,c),$
$(a,b)(b,c)(c,d),$
$(b,c)(c,d)(d,a),$
$(c,d)(d,a)(a,b) \ \}$

(b)

Figure 3.13: An Example of 1-Connected Chain Set

Certificate (a, b) needs to be stored in $D'.a$ and $D'.b$. We can repeat this process for each certificate to find the dispersal as follows:

$$\{D.a = \{(a,b),(b,c)\}, \quad D.b = \{(c,d),(d,a)\},$$
$$D.c = \{(a,b)\}, \quad D.d = \{(c,d)\} \quad \}$$

This is also an optimal dispersal of this 1-connected chain set.

To extend this solution for 1-connected chain set to k-connected chain set, we need to define an *intersection* set of a certificate. An intersection set of a certificate (u, v) in a chain set CS is a set of nodes that appear both in the set of sources and the set of destinations of the chains that contain (u, v). For certificate (a, b) in Fig. 3.13(b), the sources of the chains that contain (a, b) are $\{a, c\}$ and the destinations of such chains are $\{b, c, d\}$. The intersection of two sets is $\{c\}$, so $\{c\}$ is the intersection set of (a, b). Algorithm 8 computes an optimal dispersal of k-connected chain sets using this intersection set.

The proof of the optimality of this algorithm is straightforward. Since this algorithm considers every possible subset of the intersection set, it is guaranteed to find the optimal location set of each certificate. By Theorem 3, the dispersal computed by this algorithm is optimal.

The time complexity of this algorithm is $O(2^k \times enp)$, where k is the tight

ALGORITHM 8 : optimal dispersal of k-connected chain sets

INPUT: a k-connected chain set CS
OUTPUT: a dispersal D of CS

STEPS:
1: **for** each node u in CS, $D.u := \{\}$
2: **for** each certificate (u, v) in CS **do**
3: **compute** the intersection set IS of (u, v)
4: **for** each subset X of IS
5: **for** each node w in CS, **if** $w \in X$ **then** $D_X.w := \{(u, v)\}$ **else** $D_X.w := \{\}$
6: **compute** the chain set S of all the chains that contain (u, v)
and their sources and destinations are not in X
7: **for** each chain from y to z in S
8: **if** $y \in IS \setminus X$ **then** add (u, v) to $D_X.z$ and remove the chain from S
9: **if** $z \in IS \setminus X$ **then** add (u, v) to $D_X.y$ and remove the chain from S
10: **run** Algorithm 6 on S and add the resulting location set of (u, v) to D_X
11: **find** D_X with the minimal cost
12: **for** each node u in CS, add $D_X.u$ to $D.u$

upper bound of the number of nodes in intersection sets of all the certificates in CS, n is the number of nodes in CS, e is the number of certificates in CS, and p is the number of chains in CS. Since there are at most k nodes in the intersection set of each certificate, there are at most 2^k subsets of the intersection set. For each subset, we run Algorithm 6, whose complexity is $O(enp)$. Therefore, the total time complexity becomes $O(2^k enp)$. Since k is a constant, the time complexity becomes $O(enp)$.

3.6 Dynamic Dispersal

In the previous sections, we discussed the concept of certificate dispersal. Algorithms in Sections 3.4 show how to compute a certificate dispersal for a "static" certificate graph, i.e. the topology of the certificate graph does not change over time. However, in many certificate systems, certificate graphs do change due to issuing new certificates, adding new users, revoking old certificates, and removing old users. To maintain the certificate dispersal of a dynamic certificate graph, the changes in the graph need to be propagated to the appropriate users.

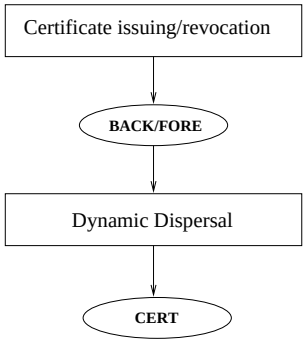


Figure 3.14: Inputs and Output of Dynamic Dispersal Protocol

Fig. 3.14 shows the inputs and output of our dynamic dispersal protocol. The dynamic dispersal protocol running at each user has two inputs `FORE` and `BACK`. `FORE` in user u is the set of the certificates that have been issued by user u, and `BACK` in user u is the set of users that have issued certificates for u. Note that the two inputs `FORE` and `BACK` in all users define the certificate graph of the system. We assume that `FORE` and `BACK` are maintained by an outside protocol that issues new certificates and revokes old ones. We also assume that `FORE` and `BACK` are always *correct* and so they are always *consistent*. For example, if at any time a certificate (u, v) is in `FORE`.u of user u, then u is in `BACK`.v of user v at the same time.

The dynamic dispersal protocol maintains a variable `CERT`.u at each user u.

At stabilization, the value of CERT.u is a outgoing spanning tree rooted at user u. Thus, by Lemma 1, values of CERTs at stabilization constitute a certificate dispersal of the system.

The dynamic dispersal protocol in user u is shown in Protocol 1 below. Protocol 1 consists of three actions.

In the first action, when the timer of user u expires, user u uses its input FORE.u to update the variable CERT.u and sends a copy of CERT.u to each user v in BACK.u. Then u updates its timer to expire after ltime time units, and the cycle repeats. For convenience, we refer to CERT.u messages that user u has sent in this action as a round of gossip. If user u does not change its CERT.u and does not observe any change in its inputs FORE.u and BACK.u, then the time period between two consecutive rounds of gossip by u is ltime time units. The value ltime is expected to be in the range of days or months.

In the second action, user u receives a certificate tree sent by a user v (where u is in BACK.v). In this case, u updates its CERT.u using its input FORE.u, and then merges its CERT.u with the received certificate tree. If the update or merge operations change CERT.u then u reduces the value of its timer to at most stime time units. Note that the value stime is in the range of minutes or hours so it is much less than the value ltime. In other words, any change in the variable CERT.u causes u to initiate its next round of gossip after no more than stime time units.

In the third action, when user u observes that its inputs BACK.u or FORE.u has changed, then user u sets its timer to be at most stime time units. This change causes u to initiate its next round of gossip after no more than stime time units.

3.6.1 Issuing certificates

When a user u issues a certificate (u, v), there are two events that need to occur. (Note that these two events occur outside the dynamic dispersal protocol.) The first

PROTOCOL 1 dynamic dispersal

```
user u

const   stime, ltime                  //stime is a short time period
                                      //ltime is a long time period
                                      //ltime is greater than stime

input   BACK        : {x| x has issued a certificate (x,u)}
        FORE        : {(u,x) | u has issued a certificate (u,x)}

var     CERT        : a certificate tree rooted at u
        tree        : a certificate tree
        timer       : 0..ltime
        v           : any user other than u

begin
     timer=0 ->         update(CERT, FORE);
                        for each user v in BACK, send CERT to v;
                        timer:=ltime

  [] rcv tree from v -> update(CERT, FORE);
                        merge(CERT, tree);
                        if CERT has changed, timer:=min(timer, stime)

  [] BACK or FORE has changed -> timer:=min(timer,stime)

end
```

event is to add (u, v) to FORE.u, and the second event is to add u to BACK.v. These events cause users u and v to execute the third action in the protocol and to reduce their timers to be at most stime time units. In stime time units, the timers in both users u and v will expire and then users u and v will execute the first action and update their CERTs and send a copy of the updated CERT to each user in their BACKs.

3.6.2 Revoking Certificates

When a user u wants to revoke a certificate (u, v) it has issued before, two events need to occur in users u and v. (Note that these two events occur outside the dynamic dispersal protocol.) The first event is to remove (u, v) from FORE.u, and the second event is to remove u from BACK.v.

When user u observes the change in FORE.u, u executes the third action and set its timer to be at most stime. When the timer expires, u will update CERT.u and send it to users in BACK.u. When user x in BACK.u receives the newly updated CERT.u from user u, x will merge it with its own CERT.x. During this merge, the revoked certificate (u, v) and any path using that certificate will be removed from CERT.x.

3.6.3 update Procedure

Procedure update(CERT,FORE) is defined as follows.

PROCEDURE 1 update(CERT, FORE)

```
INPUT: a certificate tree CERT rooted at u and
       a set of certificates FORE issued by u
OUTPUT: a certificate tree CERT rooted at u

var tmp: a certificate tree rooted at u

begin

    add all the valid certificates in FORE to tmp;
    while there is a valid certificate (x,y) in CERT where
       x != u,
       x is in tmp, and
       y is not in tmp
    do add (x,y) to tmp;
    CERT:=tmp;

end
```

It is convenient to explain this procedure by an example. Consider user a where FORE.a in user a contains one certificate (a, b) and CERT.a contains two certificates $(a, b), (b, c)$ as shown in Fig. 3.15(a). When user a issues a new certificate (a, c), FORE.a changes into $\{(a, b), (a, c)\}$. This change causes user a to execute its third action and then after stime time units to execute its first action. In the first action, procedure update(CERT.a,FORE.a) is executed. First, all the certificates in FORE.a are added to a certificate tree tmp and tmp becomes $\{(a, b), (a, c)\}$. Certificate (b, c) cannot be added to tmp because user c is already in tmp. In the last step, tmp is copied to CERT.a, and CERT.a becomes $\{(a, b), (a, c)\}$ as shown in Fig. 3.15(b).

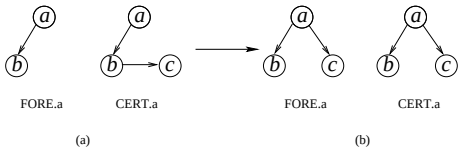


Figure 3.15: update of CERT.a due to change in FORE.a

3.6.4 merge Procedure

Procedure merge(CERT,tree) is defined as follows.

It is convenient to explain this procedure by an example. Consider user a where FORE.a contains two certificate $(a, b), (a, c)$ and CERT.a contains three certificates $(a, b), (a, c), (b, d)$ as shown in Fig. 3.16(a). When user b revokes certificate (b, d), FORE.b changes into $\{(b, c)\}$. This change causes user b to execute its third action and after stime time units to execute its first action. In the first action, user b updates its CERT.b to be $\{(b, c)\}$. User a still does not know about this revocation, so CERT.a remains the same as shown in Fig. 3.16(a). After stime time units, user b sends a copy of its CERT.b to user a. When user a receives the certificate tree $\{(b, c)\}$, user a executes its second action, and procedure merge(CERT.a,tree) is executed with CERT.a and the received tree $\{(b, c)\}$. Procedure merge(CERT.a,tree) first

PROCEDURE 2 merge(CERT, tree)

```
INPUT: a certificate tree CERT rooted at u and
       a certificate tree ‘‘tree’’ rooted at t, where
       t != u
OUTPUT: a certificate tree CERT

begin

    if CERT has a certificate (u,t) ->
        remove from CERT the subtree rooted at t, if any;
        remove from tree every subtree rooted at a node, other than t,
        that occurs in CERT;
        while tree has a valid certificate (x,y) where
              x is in CERT and
              y is not in CERT
        do add y and certificate (x,y) to CERT;
    [] CERT has no certificate (u,t) ->
        skip
    fi

end
```

checks if there is certificate (a, b) in CERT.a. There is certificate (a, b), so the subtree rooted at user b, (b, d) in CERT.a is removed from CERT.a. Then, certificate (b, c) is considered, but is not added to CERT.a because c is already in CERT.a. In result, CERT.a becomes $\{(a, b), (a, c)\}$ as shown in Fig. 3.16(b).

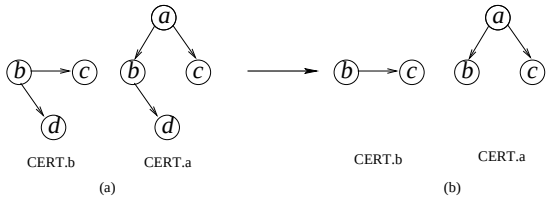


Figure 3.16: merge of CERT.a due to change in CERT.b

3.6.5 Stabilization of Dynamic Dispersal

The dynamic dispersal algorithm in Section 3.6 is based on a message passing model. In [21], it is shown to be hard to design stabilizing protocols in the traditional message passing model where there are channels between users. In this paper, we use a non-conventional model of communication. A state consists of the values of timer and `CERT` of all the users in the system. As mentioned in above, we assume that `FORE` and `BACK` of each user remain correct and consistent in every state. In one state transition, only one user can execute its first action. Furthermore, in the same transition, each user v in `BACK`.u receives the same copy of this message and executes its second action. In other words, we have no messages in transit, so there is no need for channels in the state description. There are two reasons that we adopted this model. First, this model allows the proofs to be easier to follow. Second, this model is sensible, given that the time it takes for the timer in each user to expire is very large compared to the time each state transition takes. `stime` is in the range of minutes and hours, and each state transition takes only milliseconds, so we can assume that no two timers expire at the same time.

For the proofs of convergence and closure, we define a *computation* to be a sequence of states of the system where along with this computation `FORE` and `BACK` of all the users remain unchanged. In the following theorems, we show that the dynamic dispersal protocol eventually stabilizes into a legitimate state, where the values of `CERT`s of all users constitute a certificate dispersal of the certificate graph of the system. Following the proof technique in [4], we show the convergence and the closure of this protocol to prove its stabilization.

Theorem 13 *(Convergence) Each computation of the dynamic dispersal protocol has a state where the value of each `CERT`.u in the protocol is an outgoing spanning tree rooted at u in the certificate graph of the protocol (as defined by the two inputs `FORE` and `BACK` of all users in the protocol).*

Proof sketch To prove that CERT.u eventually becomes an outgoing spanning tree rooted at node u of the certificate graph G, we first prove that CERT.u eventually becomes a tree rooted at u, and then prove that every node that is reachable from u in G is reachable in CERT.u.

There are two procedures, `update(CERT.u,FORE.u)` and `merge(CERT.u,tree)`, that can change CERT.u. The procedure `update(CERT.u,FORE.u)` constructs a tree by starting from the certificates in FORE.u. All the certificates in FORE.u are issued by user u, so the resulting tree from `update(CERT.u,FORE.u)` is rooted at u. Similarly, the procedure `merge(CERT.u,tree)` adds certificates in the received `tree` to CERT.u, a certificate tree rooted at u. Therefore, the resulting tree from `merge(CERT.u,tree)` is also rooted at u. Based on these observations, after a state transition in this computation, CERT.u in user u becomes a tree rooted at u.

Now we prove that CERT.u is an outgoing spanning tree, i.e. any node that is reachable from node u in G is also CERT.u. Assume that there is a path from u to another node v in G, $(u, u_1)(u_1, u_2) \cdots (u_k, v)$. Node u_k has the certificate (u_k, v) in its FORE, so the certificate (u_k, v) is in its CERT. Node u_k sends its CERT periodically to node u_{k-1}, so node u_{k-1} will have a path from itself to node v in its CERT. Repeatedly, each node on the path will send its CERT to the previous node in the path and node u will have a path from itself to node v in its CERT. Therefore, every node v that is reachable from node u in G is also reachable in CERT.u. ■

Note that our dynamic dispersal protocol is different from stabilizing spanning tree algorithms. The spanning tree algorithms in [15, 3, 11] build a single spanning tree for the whole system that covers every process in the system, and build one tree rooted at a special process (usually referred as a leader). Each process in these algorithms stores the parent node identifier, the distance from the root, and possibly the root identifier. On the other hand, our dynamic dispersal protocol stores an outgoing spanning tree in each user, which does not necessarily cover every

user in the system. Also, in our dynamic dispersal protocol, there is no leader, and each user u maintains an outgoing spanning tree rooted at u.

Theorem 14 *(Closure) Executing any step of the dynamic dispersal protocol starting from a state, where the value of each variable* `CERT`*.u in the protocol is an outgoing spanning tree rooted at u, leaves the values of all* `CERT` *variables unchanged.*

Proof sketch In a computation, the inputs `BACK` and `FORE` remain unchanged. Therefore, only two types of steps can be executed: time propagation and the first action. Time propagation cannot change the value of `CERT`. When the time propagation causes the timer in user u to expire, the first action in the dynamic dispersal protocol will be executed. When the timer expires, user u updates its `CERT`.u with `FORE`.u, but `CERT`.u remains the same since `FORE`.u remains unchanged. Now user u sends a copy of its `CERT`.u to each user v in `BACK`.u. User v receives a tree and merge it with its own `CERT`.v. Since `CERT`.u is the same, `merge(CERT,tree)` will not change `CERT`.v. Therefore, when the certificate graph of the system does not change, `CERT`.u in each user u, an outgoing spanning tree rooted at u, remains unchanged. ■

3.6.6 Time Complexity

In this section, we compute the time that takes to bring the system to stabilization in terms of the timer `ltime`. Note that each state transition is triggered by a timer expiration in a user, so any user will execute the first action of dynamic dispersal algorithm at least once in `ltime` time units. Also, the time that takes for a state transition is very small compared to `ltime`. Therefore, in `ltime` time units, we can assume that all users have executed the first action at least once.

Theorem 15 *In each computation of the dynamic dispersal protocol, the protocol*

reaches a legitimate state in at most T time units, where

$$T = \texttt{ltime} \times (2p - 1)$$

, where p is the length of the longest path in the certificate graph.

Proof sketch A legitimate state of the dynamic dispersal protocol is one where the value of CERT.u of every user u in the system is an outgoing spanning tree rooted at u.

Consider a certificate (x, y) that is not in the certificate graph, but in some CERT.u of user u in the beginning of the computation. This certificate disappears from CERT of any user in the system in $\texttt{ltime} \times p$. After the first ltime time units in the computation, user x updates CERT.x with FORE.x and remove the certificate (x, y) from CERT.x, if there was (x, y) in CERT.x. After the second ltime time units, any user in BACK.x receives CERT.x and removes the certificate (x, y) from its CERT, if there was (x, y) in its CERT. In other words, any user that had (x, y) in the second level of the tree in CERT removes (x, y) from its CERT. The cycle repeats, and after $(\texttt{ltime} \times p)$, any user that had (x, y) in its CERT removes (x, y) from its CERT.

Consider a certificate (v, w) that is in every possible reach tree rooted at some user u in the certificate graph, but not in CERT.u in the beginning of the computation. After the first ltime time units in the computation, user v updates CERT.v with FORE.v and add the certificate (v, w) to CERT.v if it was not in CERT.v already. For the next $(\texttt{ltime} \times (p-1))$ time units, a user in BACK.v may have node w in its CERT through a incorrect certificate and not add (v, w) to its CERT. However, any incorrect certificate will be removed from CERT of any user in $(\texttt{ltime} \times p)$ time units as shown above. Therefore, after $(\texttt{ltime} \times (p + 1))$ time units since the beginning of the computation, any user in BACK.v adds (v, w) to its CERT, if it was not there already. In other words, any user that should have (v, w) in the

second level of the tree in CERT adds (v, w) to its CERT. The cycle repeats, and after $(\texttt{ltime} \times (2p - 1))$ time units, any user that should have (v, w) in its CERT adds (v, w) to its CERT.

As shown above, in $(\texttt{ltime} \times (2p-1))$ time units, any certificate that is not in the certificate graph disappears from CERT of every user, and any certificate that is in every possible reach tree of user u appears in CERT.u. Therefore, in $(\texttt{ltime} \times (2p-1))$, CERT.u becomes an outgoing spanning tree rooted at u.

■

We believe that the upper bound on the convergence span described in Theorem 15 is quite loose. It is an interesting problem to compute a tight upper bound of the convergence span.

3.6.7 Dispersal in Client/Server Systems

This dynamic dispersal protocol is useful in any dynamic certificate systems. Consider a client/server system, where there are much fewer servers than clients in the system. We can run the dynamic dispersal protocol among the servers and let any server issue a certificate for a client. Each server will have an outgoing spanning tree in its CERT, so each server will be able to find a certificate chain from itself to any client that has a certificate issued by an authenticated server.

For example, many coffee shops offer free Internet connection for their customers. To prevent free-riders that are not customers, coffee shops may require the customers to register. For convenience, a customer needs to register only once at any coffee shop (the coffee shop issues a certificate for the customer), and the customer can use the free connection at all coffee shops that are participating in this membership without logging in or getting temporary authorization each time he or she goes to a coffee shop, since any coffee shop has a certificate chain from itself to the customer. The authentication using the certificate chain does not require

any interaction with the customer, so once the customer registers to get a certificate from one coffee shop, the customer does not need to know how he or she gets authenticated and authorized for the Internet connection.

Also, this client/server system can help two clients authenticate each other. A client $c1$ has issued a certificate for a server $s1$ and $s1$ issued a certificate for $c1$. A client $c2$ has issued a certificate for a server $s2$ and $s2$ issued a certificate for $c2$. When client $c1$ wants to securely communicate with client $c2$, client $c1$ can ask server $s1$ for a certificate chain from $s1$ to $s2$ and use the chain and the certificates $(c1, s1)$ and $(s2, c2)$ to find the public key of client $c2$.

A hierarchical certificate authorities used in Lotus Notes [32] is a special case of such client/server system. In a system with a hierarchical certificate authorities, the certificate graph between certificate authorities constitutes a star graph, where the root certificate authority has issued a certificate for each non-root certificate authority and each non-root certificate authority has issued a certificate for the root certificate authority. In such a system, when a client $c1$ who has issued a certificate for a certificate authority $ca1$ wants to securely communicate with another client $c2$ who has issued a certificate for a certificate authority $ca2$, $c1$ can contact $ca1$ for certificates $(ca1, root)(root, ca2)$. In Lotus Notes, $ca1$ also finds the certificate $(ca2, c2)$ from $ca2$ so that $c1$ can use the public key of $c2$ safely without communicating with $c2$.

Chapter 4

Vulnerability Analysis

The certificates issued by different users in a system can be represented by a directed graph, called the *certificate graph* of the system. Each node u in the certificate graph represents a user u with the corresponding public key $b.u$ and private key $r.u$ in the system. If a user has more than one public key, then the user will be represented by several nodes in the graph, one node for each public and private key pair. Each directed edge from node u to node v in the certificate graph represents a certificate $\langle u, v, b.v \rangle r.u$. A certificate chain from a node u to a node v is a simple path from node u to node v in a certificate graph. For nodes u and v in a certificate graph G, if u wishes to securely send messages to v, then u seeks a path from u to v in G. (There are systems where u seeks a set of paths from u to v, which will be discussed in Section 4.7.)

In a certificate graph, two types of damage can occur when the private key $r.u$ of a node u is revealed to an adversary: explicit and implicit. The explicit damage is that the adversary can impersonate node u to other nodes until it is known to other nodes that the private key $r.u$ of u is revealed to the adversary. The implicit damage is that the adversary can impersonate nodes other than u to other nodes in the system by signing forged certificates with the revealed private key $r.u$ of node

u.

As an example, consider the certificate graph in Fig. 4.1. If node a wishes to send a secure message to node g in this certificate graph, then node a needs to find a certificate chain from node a to node g. In the certificate graph in Fig. 4.1, there is one certificate chain from node a to node g, $(a, d), (d, e), (e, g)$.

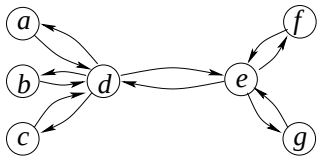


Figure 4.1: An example of a certificate graph

Assume that the private key $r.d$ of node d is revealed to an adversary. The adversary can encrypt and decrypt messages using $r.d$ to impersonate node d to any other node in the graph. This impersonation of node d is explicit damage. Assume that node a does not know that $r.d$ is revealed. The adversary can create a new public and private key pair, $b.g'$ and $r.g'$, and sign a forged certificate $\langle d, g, b.g' \rangle r.d$ with the revealed private key $r.d$ of node d. Node g' in Fig. 4.1 denotes the impersonated user g with the public and private key pair $b.g'$ and $r.g'$ created by the adversary, and the dotted edge (d, g') denotes the forged certificate $\langle d, g, b.g' \rangle r.d$. The certificate chain $(a, d), (d, g')$ presents to node a the public key $b.g'$ created by the adversary as if it belonged to user g. This impersonation of node g is implicit damage.

The explicit and implicit damage that can be brought into a certificate graph when the private key of a node u is revealed to an adversary is called the *vulnerability* of node u. For example, if the private key $r.d$ of node d in Fig. 4.1 is revealed to an adversary, then the adversary can impersonate node d to all other nodes in the graph without forging any certificates. In addition to impersonating node d, the adversary can impersonate nodes a, b, c to nodes e, f, g by signing forged certificates (d, a'),

(d, b'), and (d, c') with the revealed private key $r.d$ of node d. Also, the adversary can impersonate nodes e, f, g to nodes a, b, c by forging certificates (d, e'), (e', f'), and (e', g').

We have identified a metric to quantify the damage from this type of attacks. We call this metric *"vulnerability"* of certificate graphs. As discussed in detail in this chapter, a metric of the vulnerability of certificate graphs is useful in answering several questions. First, how to determine which certificate graphs are less vulnerable and which ones are more vulnerable. Second, how to determine which criteria for accepting public keys from certificate chains are better. Third, how to balance between the resilience against impersonation attacks and storage cost. There have been intuitive answers to these questions, such as short certificate chains and many independent certificate chains are preferable than long and dependent ones. The vulnerability metric quantifies how effectively these intuitive answers work to reduce vulnerability.

In the following sections, we formally define the vulnerability metrics of nodes and of certificate graphs, and present theorems that show vulnerabilities of several certificate graphs with different requirements. Also, we present three algorithms to compute the vulnerability of an arbitrary certificate graph. Using these algorithms, we investigate the effect of graph topology, certificate dispersal, and acceptance criteria on the vulnerability of certificate graphs. Then we discuss the vulnerability when many private keys are revealed to an adversary. We present a brief summary of related work and end with concluding remarks.

4.1 Vulnerability of Certificate Graphs

Let G be a certificate graph and d be a node in G. We assume that each node in G stores the certificates it issues, and each node accepts all public keys in a certificate chain as long as each certificate in the chain is verified. (We will discuss in Section 4.6

about the case when each node requires more than one chain to accept the public key.) Assume that the private key $r.d$ of node d is revealed to an adversary. The adversary can use the revealed private key to encrypt and decrypt any messages as if the adversary were node d, and so it can impersonate node d to all other nodes from which there are certificate chains to d. Also, the adversary can use $r.d$ to impersonate a node dst, other than d, to another node src, also other than d in G, by performing the following three steps.

i. The adversary creates a private key $r.dst'$ and its corresponding public key $b.dst'$. Later, the adversary will pretend that these keys are the public and private keys of node dst.

ii. The adversary uses the revealed private key $r.d$ of node d to issue a forged certificate $\langle d, dst, b.dst' \rangle r.d$. This forged certificate is denoted (d, dst').

iii. The adversary provides node src with the certificate chain that consists of a chain of correct certificates from src to d and the forged certificate (d, dst'). From this chain, node src can wrongly deduce that the public key $b.dst'$ created by the adversary is the public key of node dst. Any message sent by the adversary that is encrypted with the matching private key $r.dst'$ will be authenticated by node src as if it were sent by node dst.

Note that this scenario of the adversary would work only if G has a certificate chain from src to d that does not contain any certificate issued by dst and G has no certificate (src, dst).

The next theorem states a necessary and sufficient condition for an adversary to impersonate node dst to another node src in a certificate graph where the private key $r.d$ of some node d is revealed to an adversary.

Theorem 16 *Let G be a certificate graph and src and dst be any two distinct nodes in G. Let d be a node in G whose private key r.d is revealed to an adversary. The*

adversary can impersonate node dst to node src if and only if $src \neq d$, *G has a certificate chain from src to d that does not contain any certificates issued by node dst, and one of the following two conditions holds.*

i. $dst = d$, *or*

ii. G has no certificate (src, dst)

Proof:

Proof for if If $dst = d$, then the adversary can use the revealed private key $r.d$ of node d to encrypt and decrypt any message as if it were node d and impersonate node dst to node src. If $dst \neq d$, then G has no certificate (src, dst), so src does not know the correct public key of dst. Now the adversary can sign a forged certificate (d, dst') with the revealed private key $r.d$ of d. There is a certificate chain from src to d that does not contain any certificates issued by dst, so the adversary can add the forged certificate (d, dst') to the correct certificate chain from src to d and present the certificate chain from src to dst' to node src. If node src does not know that the private key of node d is revealed to the adversary, then src will not notice that the certificate (d, dst') is forged and accept the public key in (d, dst') as the valid public key of dst.

Proof for only if In order to prove the only if part, we prove the contraposition. If any of the following three conditions holds, then the adversary cannot impersonate dst to src:

i. $src = d$

ii. G has no certificate chain from src to d that does not contain any certificate issued by dst.

iii. $dst \neq d$ and G has certificate (src, dst).

First, assume $src = d$. In this case, src will not accept any forged certificate including a new public key created by the adversary, since src stores all the certificates it issued.

Second, assume that G has no certificate chain from src to d that does not contain any certificate issued by dst. If G has no certificate chain from src to d that does not contain any certificate issued by dst, there are two possible cases to consider. In the first case, G has no certificate chain from src to d. In the second case, G has at least one certificate chain from src to d, but every such chain from src to d contains a certificate issued by dst. In the first case, the adversary cannot create a certificate chain from src to dst', because G has no certificate chain from src to d to which the adversary can add a forged certificate (d, dst'). So the adversary cannot impersonate dst to src. In the second case, src will verify the public key of node dst in the process of validating the certificate chain from src to d, and will notice that the identity of dst is repeated twice in the certificate chain and reject the public key of dst'. In both cases, the adversary cannot impersonate dst to src.

Third, assume $d \neq dst$ and G has certificate (src, dst). If src has issued the certificate (src, dst), then src already knows the correct public key of dst, so it will not accept any other public key created by the adversary as a valid public key of dst. Hence, the adversary cannot impersonate dst to src. This completes the proof for the only if part. ■

Let G be a certificate graph and d be a node in G. Assume that the private key $r.d$ of node d is revealed to an adversary. The *vulnerability of node d*, denoted $V(d)$, is the number of node pairs (src, dst) where the adversary can impersonate node dst to node src divided by the number of node pairs (src, dst) where $src \neq dst$ and $src \neq d$ in G. More formally,

$$V(d) = \frac{|IMP(d)|}{(n-1)^2},$$

where $IMP(d) = \{(src, dst)|$ the adversary can impersonate dst to src using $r.d\}$ and n is the number of nodes in G.

The following theorem gives tight upper and lower bounds on the vulnerability of a node in a certificate graph.

Theorem 17 *For a node d in any certificate graph G, we have*

$$1 \geq V(d) \geq$$

$$\frac{|\{src|G \text{ has a certificate chain from } src \text{ to } d\}|}{(n-1)^2}$$

Proof: The most number of node pairs (src, dst) where the adversary can impersonate dst to src is the total number of node pairs (src, dst) where $src \neq dst$ and $src \neq d$, which is $(n-1)^2$. Therefore, the upper bound of $V(d)$ is 1. Also, since the adversary knows $r.d$, the adversary can always impersonate node d to every node that has a certificate chain from itself to d. (This is the scope of explicit damage.) Therefore, the number of node pairs (src, d) where G has a certificate chain from src to d divided by $(n-1)^2$ is the lower bound. ■

The following lemmas show that the bounds shown in the above theorem are tight.

Lemma 6 *There exists node d in some certificate graph G, where*

$$V(d) = 1$$

Proof: Consider the certificate graph in Fig. 4.2. When the private key of the center node is revealed to an adversary, the adversary can impersonate any node dst to any other node src, where src is not the center node. There are 8 nodes that can be src, and for each src node among them, there are 8 other nodes that can be impersonated to src. Therefore, the number of node pairs (src, dst) where

the adversary can impersonate dst to src is $8 \times 8 = 64$, and $n = 9$. Therefore, the vulnerability of the center node is 1. ■

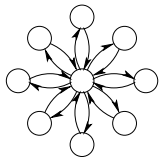


Figure 4.2: The $(8, 1)$-star certificate graph

Lemma 7 *There exists node d in some certificate graph G, where*

$$V(d) = \frac{|\{src | G \text{ has a certificate chain from } src \text{ to } d\}|}{(n-1)^2}$$

Proof: In the certificate graph in Fig. 4.3, every node has issued certificates to all other nodes in the graph. If the private key of node c is revealed to an adversary, the adversary can impersonate only node c to nodes a and b, since node a already knows the correct public key of node b in the certificate (a, b) and node b knows the correct public key of node a in the certificate (b, a). So the vulnerability of node c is $\frac{2}{2^2}=\frac{1}{2}$, which meets the lower bound. In fact, the vulnerability of any node in a fully connected certificate graph meets the lower bound. ■

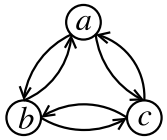


Figure 4.3: An example of fully connected certificate graph

Let G be a certificate graph, then the vulnerability of graph G, denoted $V(G)$, is defined as follows:

$$V(G) = \max_{d \in G} V(d)$$

4.2 Vulnerability of Special Certificate Graphs

In this section, we give three theorems that show the vulnerability of three special classes of certificate graphs: n-loops, (m, k)-stars, and (d, h)-trees. In many certificate systems, for example PGP, certificate graphs are not planned in advance and certainly not designed. Rather, they are developed in an ad-hoc manner depending on which users decide to issue certificates for which other users. However, if we do have the luxury of planning and designing certificate graphs, then we can choose the best among these special classes according to the system requirements. n-loop certificate graphs are useful when the certificate graph needs to be strongly-connected but the number of certificates needs to be minimized. (m, k)-star certificate graphs are useful when a trusted certificate authority (center node) is available. (d, h)-tree certificate graphs are useful in hierarchical systems.

The following three theorems compute the vulnerabilities of three special classes of certificate graphs. The theorems show that n-loop certificate graphs are less vulnerable than $(m, 2)$-star certificate graphs for $n \geq 4$. On the other hand, $(2, h)$-tree certificate graphs are less vulnerable than n-loop certificate graphs for $n > 10$. The comparison results are discussed in more detail in the end of this section.

An n-loop certificate graph is a certificate graph that has n nodes arranged in a unidirectional ring. Fig. 4.4 shows the 8-loop certificate graph.

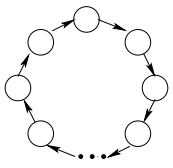


Figure 4.4: The 8-loop certificate graph

Theorem 18 *The vulnerability of an n-loop certificate graph is* $1 - \frac{n-2}{2(n-1)}$.

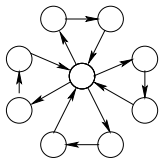


Figure 4.5: The $(4, 2)$-star certificate graph

Proof: Label each node $0 \cdots n-1$. Assume that the private key of node j is revealed to an adversary. The adversary can impersonate node k to node i if $k = j$, or if $\nexists(i, k)$ and there is a path from node i to node j that does not contain node k. Therefore, to node $j-1$, the adversary can impersonate nodes $j, j+_n 1, \cdots, j+_n (n-2)$. To node $j-2$, the adversary can impersonate nodes $j, j+_n 1, \cdots, j+_n (n-3)$. After considering each node, the number of (src, dst) pairs in which the adversary can impersonate node dst to node src is $\frac{n(n-1)}{2}$. The vulnerability of node j is $\frac{n(n-1)}{2(n-1)(n-1)} = 1 - \frac{n-2}{2(n-1)}$. This holds for any node j in this graph, so the vulnerability of an n-loop certificate graph is $1 - \frac{n-2}{2(n-1)}$. ■

An (m, k)-star certificate graph is a certificate graph that consists of m unidirectional rings that share one center node and each ring has k unshared nodes. Fig. 4.5 shows the $(4, 2)$-star certificate graph.

Theorem 19 *The vulnerability of an (m, k)-star certificate graph is $1 - \frac{k-1}{2mk}$.*

Proof: The vulnerability of a graph is the maximum vulnerability of every node in the graph. In an (m, k)-star certificate graph, the center node has the highest vulnerability. Now let us compute the vulnerability of the center node. Label the k nodes in a satellite ring from $1 \cdots k$ and the center node as node 0. There is an edge from node i to node $i +_{k+1} 1$, where $0 \leq i \leq k$. When the private key of the center node is revealed to an adversary, the adversary can impersonate to node 1 any node in the graph except for the nodes $2 \cdots k$ in the same satellite ring. To node 2, the adversary can impersonate any node in the graph except for the nodes $3 \cdots k$ in the same satellite ring. As a result, the adversary can impersonate $\sum_{i=1}^{k}(mk - (k-i))$

pairs for each satellite ring. So the vulnerability of the center node is

$$\begin{aligned} V(center) &= \frac{1}{(mk)^2}(m\sum_{i=1}^{k}(mk-(k-i))) \\ &= \frac{1}{(mk)^2}\left(mk(mk-k)+\frac{mk(k+1)}{2}\right) \\ &= \frac{1}{mk}\left((mk-k)+\frac{k+1}{2}\right) \\ &= \frac{2mk-2k+k+1}{2mk} \\ &= \frac{2mk-k+1}{2mk} \\ &= 1-\frac{k-1}{2mk} \end{aligned}$$

Therefore, the vulnerability of an (m,k)-star certificate graph is $1-\frac{k-1}{2mk}$. ■

A (d,h)-tree certificate graph is a complete tree certificate graph with degree d and height h, where there is an edge from each parent node to each of its children nodes and an edge from each child node to its parent node. Fig. 4.6 is an example of a (d,h)-tree certificate graph, where $d=3$ and $h=2$.

Theorem 20 *The vulnerability of a (d,h)-tree certificate graph is $1-\frac{d^{h+1}+hd^{h+1}}{(d-1)(n-1)^2}-\frac{d}{(d-1)(n-1)}-\frac{d^2}{(d-1)(n-1)^2}$, approximately $1-\frac{h}{d^h}$.*

Proof: The vulnerability of a graph is the maximum of vulnerability of all nodes in the graph. In a (d,h)-tree certificate graph, the root node has the highest vulnerability. The vulnerability of the root node can be computed as follows. Consider a node i in level h. When the private key of the root node is revealed to an adversary, the adversary can impersonate any node to node i except the $(h-1)$ nodes on the certificate chain from node i to the root node. On the other hand, for a node j in level $h-1$, the adversary can impersonate any node to node j except its d children nodes and the $(h-2)$ nodes on the certificate chain from node j to the root node. So, the adversary can impersonate $(n-1-(h-2+d))$ nodes to node j. Similarly,

for a node in level l, where $l < h$, the adversary can impersonate $(n-1-(l-1+d))$ nodes to the node. As a result, the vulnerability of the root node is :

$$
\begin{aligned}
V(root) &= \frac{1}{(n-1)^2}\left(\sum_{i=1}^{h-1} d^i(n-1-(i-1+d)) + d^h(n-1-(h-1))\right) \\
&= \frac{1}{(n-1)^2}\left((n-d)\sum_{i=1}^{h-1} d^i - \sum_{i=1}^{h-1} id^i + (n-h)d^h\right) \\
&= \frac{1}{(n-1)^2}\left(n\sum_{i=1}^{h} d^i - \sum_{i=1}^{h-1} d^{i+1} - \sum_{i=1}^{h} id^i\right) \\
&= \frac{1}{(n-1)^2}\left(n(n-1) - \frac{d^2(d^{h-1}-1)}{d-1} - \frac{hd^{h+1}-n+1}{d-1}\right) \\
&= \frac{n}{n-1} - \frac{d^{h+1}-d^2+hd^{h+1}-n+1}{(d-1)(n-1)^2} \\
&= 1 - \frac{1}{n-1} - \frac{d^{h+1}-d^2+hd^{h+1}-n+1}{(d-1)(n-1)^2} \\
&= 1 - \frac{d^{h+1}+hd^{h+1}}{(d-1)(n-1)^2} - \frac{1}{n-1} - \frac{d^2}{(d-1)(n-1)^2} - \frac{1}{(d-1)(n-1)} \\
&= 1 - \frac{d^{h+1}+hd^{h+1}}{(d-1)(n-1)^2} - \frac{d-1+1}{(d-1)(n-1)} - \frac{d^2}{(d-1)(n-1)^2} \\
&= 1 - \frac{d^{h+1}+hd^{h+1}}{(d-1)(n-1)^2} - \frac{d}{(d-1)(n-1)} - \frac{d^2}{(d-1)(n-1)^2} \\
&\simeq \{\text{since } n \text{ is large}\}\, 1 - \frac{d^{h+1}+hd^{h+1}}{(d-1)(n-1)^2} \\
&\simeq \{\text{since } n = \frac{d^{h+1}-1}{d-1} \simeq d^h\}\, 1 - \frac{d^{h+1}(1+h)}{(d-1)(d^h)^2} \\
&\simeq \{\text{since } \frac{h+1}{d-1} \simeq \frac{h}{d}\}\, 1 - \frac{hd^{h+1}}{d(d^h)^2} \\
&= 1 - \frac{h}{d^h}
\end{aligned}
$$

Therefore, the vulnerability of a (d,h)-tree certificate graph is approximately $1-\frac{h}{d^h}$.

■

Fig. 4.7 shows the vulnerabilities of three special certificate graphs, n-loops,

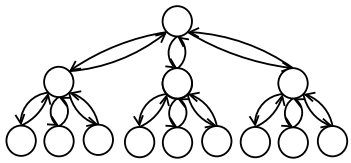


Figure 4.6: The $(3, 2)$-tree certificate graph

$(m, 2)$-stars, and $(2, h)$-trees as functions of the number of nodes in each graph. From this graph, it is clear that n-loops are less vulnerable than $(m, 2)$-stars and $(2, h)$-trees. This metric of vulnerability can be used to show which certificate graph is less vulnerable.

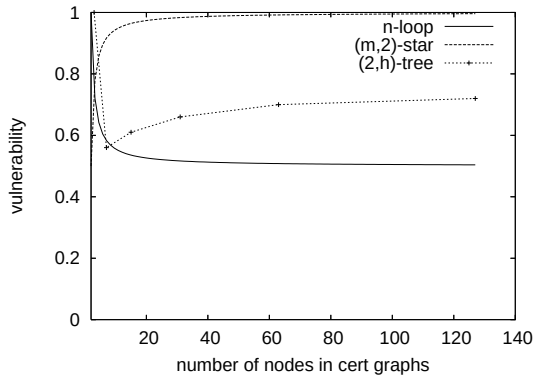


Figure 4.7: Comparison of three special graphs

4.3 Vulnerability of Arbitrary Certificate Graphs

In the previous section we computed the vulnerability of three special classes of certificate graphs. We now present Algorithm 9 that computes the vulnerability of an arbitrary certificate graph.

By Theorem 16, if G has a path from node src to node d that does not contain node dst, then the adversary can impersonate dst to src when the private key of node d is revealed to it (and G has no certificate (src, dst)). As mentioned

in Section 4.1, we assume that each node only stores the certificates it issued and accepts all the public keys in the presented certificate chain as long as each certificate in the chain is verified. To find every node src that has a path to node d that does not contain node dst, Algorithm 9 removes node dst and its incoming and outgoing edges from G and sees which nodes are still connected to d. Consider the example certificate graph G in Fig. 4.1. In Fig. 4.8(a), node a and its incoming and outgoing edges are removed from G. There are paths from nodes b, c, e, f, g to node d in Fig. 4.8(a). Therefore, if the private key of node d is revealed to an adversary, then the adversary can impersonate node a to nodes b, c, e, f, g. On the other hand, without node e and its incoming and outgoing edges, there are no paths from nodes f, g to node d as shown in Fig. 4.8(b). Therefore, when the private key of d is revealed to an adversary, the adversary cannot impersonate node e to nodes f, g.

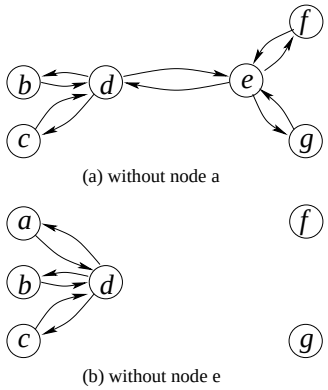


Figure 4.8: Computing vulnerability of the example graph

For a given certificate graph G, Algorithm 9 computes a transitive closure C_{dst} without using any incoming and outgoing edges of node dst for each node dst in G (lines 3-4). C_{dst} contains an edge (src, d) if and only if there is a path from src to d that does not contain dst and G has no certificate (src, dst) (line 5). In other words, if there is an edge (src, d) in C_{dst}, then an adversary can impersonate dst to src when the private key of node d is revealed to the adversary.

ALGORITHM 9 : Vulnerability of a certificate graph

INPUT: a certificate graph G with n nodes
OUTPUT: vulnerability of G

STEPS:
1: **for** $dst = 0$ **to** $n - 1$
2: $C_{dst} := G$
3: **remove** all the incoming and outgoing edges of node dst from C_{dst}
4: C_{dst} := transitive closure of C_{dst}
5: **if** G has an edge (src, dst) for any node src, **then remove** (src, dst) from C_{dst}
6: **endfor**
7: C := transitive closure of G
8: **for** $d = 0$ **to** $n - 1$
9: $V(d) := \sum_{dst \in G}$(the in-degree of node d in C_{dst}) + the in-degree of node d in C
10: **endfor**
11: **return** $\max_{d \in G} \frac{V(d)}{(n-1)^2}$

To compute the vulnerability of a node d in G, Algorithm 9 finds all the node pairs (src, dst) in G such that G has a path from src to d that does not contain dst and has no certificate (src, dst). For each node dst in G, the in-degree of node d in the transitive closure C_{dst} is the number of node pairs (src, dst) in G that satisfies the condition. So the sum of the in-degree of node d in the transitive closure C_{dst} for each node dst in G shows the scope of the implicit damage of the revealed private key of node d.

In the example certificate graph G in Fig. 4.1, when the private key of d is revealed to an adversary, the adversary can impersonate node d to any other user in G. To compute this explicit damage of the revealed private key of node d, Algorithm 9 also computes a transitive closure C of G (line 7). C contains an edge (src, d) if and only if there is a path from src to d in G. In other words, if there is an

edge (src, d) in C, then the adversary can impersonate d to src using the revealed private key of node d. Therefore, the in-degree of node d in the transitive closure C of G shows the scope of the explicit damage of the revealed private key of node d.

Using these transitive closures, Algorithm 9 computes the vulnerability of each node d in a given certificate graph G, and then returns the maximum as the vulnerability of the certificate graph.

In this algorithm, the most expensive step is line 4. The cost of computing a transitive closure of a certificate graph with n nodes is $O(n^3)$, and we need to compute $(n+1)$ transitive closures. Therefore, the complexity of this algorithm is $O(n^4)$.

4.4 Effect of Topology on Vulnerability

The vulnerability of a certificate graph is affected by the topology of the graph. For example, the $(4,2)$-star certificate graph in Fig. 4.5 has vulnerability $\frac{15}{16}$, whereas the $(8,1)$-star certificate graph has vulnerability 1. Therefore, these two certificate graphs, despite having the same number of nodes and the same connectivity, have different vulnerabilities.

In Fig. 4.9, we show the effect of topology on vulnerability of star certificate graphs. Theorem 19 gives the vulnerability of (m,k)-star certificate graphs. However, if we keep the same number of nodes in the star certificate graph but change the value of k, not every satellite ring can have exactly k nodes. We put k nodes in as many rings as possible, and leave the remaining nodes in the last ring. We ran Algorithm 9 on the star certificate graphs with 100 nodes where k, the maximum number of nodes in each satellite ring, changes from 1 to 99. Fig. 4.9 shows that the vulnerability decreases as k increases.

In Fig. 4.10, we show the effect of topologies on vulnerability of tree certificate graphs. Theorem 20 gives the vulnerability of (d,h)-tree certificate graphs. However,

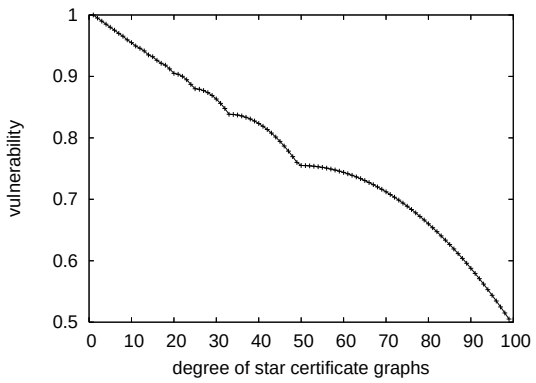


Figure 4.9: Vulnerability of Star Certificate Graphs

if we keep the same number of nodes in the certificate graph but change the value of d, the resulting tree may not be complete. In those trees, we pack the leaf nodes to the left. We ran Algorithm 9 on the tree certificate graphs with 100 nodes where d, the degree of tree, changes from 2 to 99. Fig. 4.10 shows that the vulnerability increases as d increases.

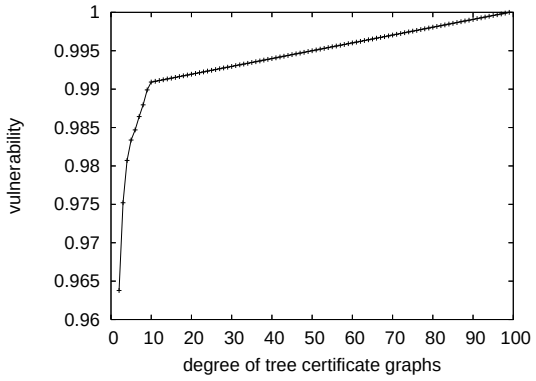


Figure 4.10: Vulnerability of Tree Certificate Graphs

4.5 Effect of Dispersal on Vulnerability

In a certificate graph where certificate chains are used to find a public key, nodes may store a few certificates in their local storage to expedite the search for a public key [22, 23, 38]. In particular, *certificate dispersal* D of a certificate graph G in Chapter 3.6 assigns a set of certificates $D.u$ to each node u so that if G has a certificate chain from node u to node v, then $D.u \cup D.v$ contains all the certificates in the certificate chain. If certificate dispersal is applied, then when a node u wishes to securely communicate with a node v, then node u will look for a public key of node v in $D.u$ first before it sends out a query to node v for more certificates. Therefore, if node u already has a certificate that has node v as the subject of the certificate, the adversary cannot impersonate node v to node u by issuing forged certificates. In other words, the vulnerability of a certificate graph is not only determined by the topology of the certificate graph, but also affected by the dispersal of the certificate graph.

As mentioned in Section 4.3, when no dispersal is deployed, if the private key of node d is revealed to an adversary, then the adversary can impersonate nodes d, e, f, g to nodes a, b, c, and impersonate nodes a, b, c, d to nodes e, f, g. However, when we assign all the certificates in an outgoing spanning tree rooted at node x to the set $D.x$, if the private key of node d is revealed to an adversary, then the adversary can impersonate only node d to all other nodes, so there can be no implicit damage to the graph. Theorem 16 is modified here to take the effect of dispersal into consideration.

Theorem 21 *Let G be a certificate graph and src and dst be any two distinct nodes in G. Let D be any dispersal of G and d be a node in G whose private key r.d is revealed to an adversary. The adversary can impersonate node dst to node src if and only if $src \neq d$, G has a certificate chain from src to d that does not contain any certificate issued by node dst, and one of the following two conditions holds.*

i. $dst = d$, *or*

ii. $D.src \not\ni (k, dst), k \in G$

Proof:

Proof for if If $dst = d$, then the adversary can use the revealed private key $r.d$ of node d to encrypt and decrypt any message as if it were node d and impersonate node dst to node src. If $dst \neq d$, then $D.src$ has no certificate (k, dst), for any node k in G, so src does not know the correct public key of dst. Now the adversary can sign a forged certificate (d, dst') with the revealed private key $r.d$ of d. There is a certificate chain from src to d that does not contain any certificates issued by dst, so the adversary can add the forged certificate (d, dst') to the correct certificate chain from src to d and present the certificate chain from src to dst' to node src. If node src does not know that the private key of node d is revealed to the adversary, then src will not notice that the certificate (d, dst') is forged and accept the public key in (d, dst') as the valid public key of dst.

Proof for only if In order to prove the only if part, we prove the contraposition. If any of the following three conditions holds, then the adversary cannot impersonate dst to src:

i. $src = d$

ii. G has no certificate chain from src to d that does not contain any certificate issued by dst.

iii. $dst \neq d$ and $D.src$ has certificate (k, dst), for some node k in G.

First, assume $src = d$. In this case, src will not accept any forged certificate including a new public key created by the adversary, since src stores all the certificates it issued.

Second, assume that G has no certificate chain from src to d that does not contain any certificate issued by dst. If G has no certificate chain from src to d

that does not contain any certificate issued by dst, there are two possible cases to consider. In the first case, G has no certificate chain from src to d. In the second case, G has at least one certificate chain from src to d, but every such chain from src to d contains a certificate issued by dst. In the first case, the adversary cannot create a certificate chain from src to dst', because G has no certificate chain from src to d to which the adversary can add a forged certificate (d, dst'). So the adversary cannot impersonate dst to src. In the second case, src will verify the public key of node dst in the process of validating the certificate chain from src to d, and will notice that the identity of dst is repeated twice in the certificate chain and reject the public key of dst'. In both cases, the adversary cannot impersonate dst to src.

Third, assume $d \neq dst$ and $D.src$ has certificate (k, dst) for some node k in G. Based on certificate (k, dst), src already knows the correct public key of dst, so it will not accept any other public key created by the adversary as a valid public key of dst. Hence, the adversary cannot impersonate dst to src. This completes the proof for the only if part. ■

Algorithm 10 shown below is modified from Algorithm 9 to include the effect of dispersal in the evaluation of vulnerability. If node src has a certificate (x, dst) due to dispersal for any user x, then no adversary can impersonate dst to x with the revealed private key of any user y. Specifically, after line 4 in Algorithm 9, the following line is added: if any $D.src$ has an edge (x, dst), then remove all the edges (src, y) from C_{dst}.

The graph in Fig. 4.11 shows how much vulnerability is reduced by the optimal certificate dispersal of tree certificate graphs. In the case of "No Dispersal", each node knows only the public keys in the certificates it issued. In the case of "With Dispersal", each node stores certificates assigned by an optimal dispersal of the certificate graph and knows the public keys in the stored certificates. The cost of certificate dispersal is defined as the average number of certificates stored in each

ALGORITHM 10 : Vulnerability with certificate dispersal

INPUT: a certificate graph G with n nodes and a dispersal D of G
OUTPUT: vulnerability of G
STEPS:
1: **for** $dst = 0$ **to** $n - 1$
2: $C_{dst} := G$
3: **remove** all the incoming and outgoing edges of node dst from C_{dst}
4: C_{dst} := transitive closure of C_{dst}
5: **if** any $D.src$ has an edge (x, dst), **then remove** all the edges (src, y) from C_{dst}
6: **endfor**
7: C := transitive closure of G
8: **for** $d = 0$ **to** $n - 1$
9: $V(d) := \sum_{dst \in G}$(the in-degree of node d in C_{dst}) + the number of edges (src, d) in C for any node src in G
10: **endfor**
11: **return** $\max_{d \in G} \frac{V(d)}{(n-1)^2}$

node. An optimal dispersal of a certificate graph is a dispersal whose cost is less than or equal to the cost of any other dispersal of the same certificate graph. The tree certificate graphs have 100 nodes and the degree changes from 2 to 99. The result without dispersal is the same as Fig. 4.10.

Note that the cost of the optimal dispersal of tree certificate graphs decreases, as shown in Fig. 4.12, whereas the vulnerability increases, as the degree of the tree increases. The x-axis of the graph in Fig. 4.12 is same as Fig. 4.11, and the y-axis shows the optimal dispersal cost. There is a clear trade-off between the vulnerability and the optimal dispersal cost of tree certificate graphs.

The trade-off between the dispersal cost and the vulnerability in general is fairly straightforward, since a higher dispersal cost means that nodes know more correct public keys, corresponding to more nodes that the adversary will not be

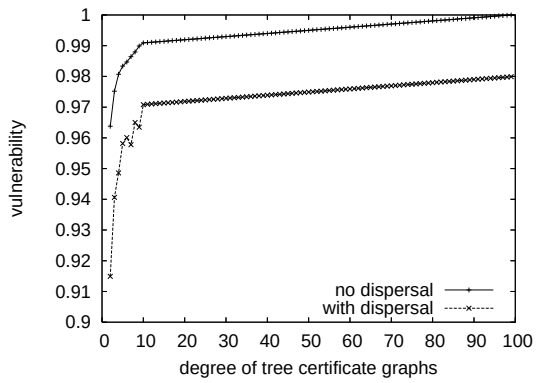


Figure 4.11: Vulnerability of Trees with Optimal Dispersal

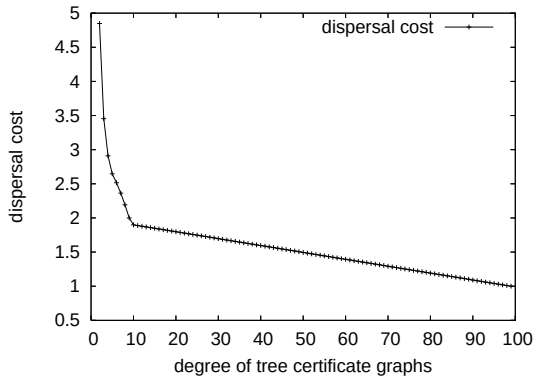


Figure 4.12: Optimal Dispersal Cost of Tree Graphs

able to impersonate. However, the trade-off between the dispersal cost of the optimal dispersal and the vulnerability shown here suggests that the certificate graph topology and the certificate dispersal algorithm must be carefully chosen to reach the right balance between the performance overhead (i.e. the size of local storage for dispersed certificates) and the resilience against attacks (i.e. the vulnerability). A graph with little vulnerability may be resource-intensive to accommodate higher dispersal cost, which is suitable for high assurance networks. On the other hand, a graph with limited storage may prefer a certificate graph topology with small dispersal cost and suffer a higher exposure to impersonation attacks.

4.6 Effect of Acceptance Criteria on Vulnerability

To reduce the implicit damage of a revealed private key of a node, many researchers proposed to use some acceptance criteria to verify the validity of the public key of the destination node of the certificate chain [5, 24, 27, 30, 33, 34, 36, 40]. Some of the results are described in Chapter 5. Most of these criteria can be modeled as a function that takes a set of certificate chains as an input and outputs a yes/no answer. A node u, who wants to find the public key of another node v, will find a set of certificate chains from u to v. Node u can give this set as an input to the acceptance criteria function, and if the output answer is yes, then the public key of node v in the certificate chain will be accepted by node u as valid. Theorem 16 is modified here to take the acceptance criteria into consideration.

Theorem 22 *Let G be a certificate graph and src and dst be any two distinct nodes in G. Let d be a node in G whose private key $r.d$ is revealed to an adversary. The adversary can impersonate node dst to node src if and only if $src \neq d$ and one of the following two conditions holds.*

i. $d = dst$ and G has a set of certificate chains from src to dst that satisfies the acceptance criteria of G, or

ii. $\nexists(src, dst)$ and the set of certificate chains where each chain in the set consists of a correct certificate chain from src to d, that does not contain any certificate issued by node dst, and a forged certificate (d, dst'), satisfies the acceptance criteria of G.

A simple acceptance criteria is to limit the length of certificate chains that can be used. For example, a node might set the value of this limit to be 6 and accept only chains that consist of 6 or fewer certificates. In fact, this acceptance criteria is implemented in the current PGP system as the parameter `CERT_ DEPTH`.

Algorithm 11 shown below computes vulnerability of certificate graphs in the case where this acceptance criteria is used. To explain Algorithm 11, we need to define the concept of k-closure.

A k-closure of a graph G is a directed graph that has the same number of nodes in G, and this graph has an edge (src, dst) if and only if there is a directed path of length at most k from src to dst in G. Note that 1-closure of G is G itself, and 0-closure of G is a graph with the same nodes in G but does not contain any edges.

Algorithm 11 takes a certificate graph G and the limit k(=CERT_DEPTH) on chain length as input and compute (k-1)-closures for each node dst, so that the adversary can add a forged certificate to the existing chain and the resulting chain will satisfy the limit k on chain length.

ALGORITHM 11 : Vulnerability with limit k on chain length

INPUT: a certificate graph G with n nodes and
a limit k(=CERT_DEPTH) on chain length
OUTPUT: vulnerability of G

STEPS:
1: **for** $dst = 0$ **to** $n - 1$
2: $C_{dst} := G$
3: **remove** all the incoming and outgoing edges of node dst from C_{dst}
4: $C_{dst} :=$ $(k-1)$-closure of C_{dst}
5: **if** G has an edge (src, dst), **then remove** (src, dst) from C_{dst}
6: **endfor**
7: $C :=$ k-closure of G
8: **for** $d = 0$ **to** $n - 1$
9: $V(d) := \sum_{dst \in G}$(the in-degree of node d in C_{dst}) + the in-degree of node d in C
10: **endfor**
11: **return** $\max_{d \in G} \frac{V(d)}{(n-1)^2}$

The graphs in Figs. 4.13-4.14 show how vulnerability changes as we apply different `CERT_DEPTH` as the limit on chain length. As `CERT_DEPTH` increases, a node can accept longer chains, and the vulnerability increases. In Fig. 4.13, each star certificate graph has 100 nodes and 10 satellite rings, and the maximum number of nodes in a satellite ring is 10. We changed the value of `CERT_DEPTH` from 1 to 11, since the longest chain that the adversary will use from the original certificate graph is 10. (The longest chain from a node in a satellite ring to the center node is 10.) After 10, the vulnerability is same as that in Fig. 4.9. For comparison, we show the vulnerability of the graph without applying `CERT_DEPTH` shown as a dotted line here.

In Fig. 4.14, each tree certificate graph has 100 nodes and the degree is 2. Since the root node has the maximum vulnerability, the longest chain that an adversary will use from the original certificate graph is from the leaf node to the root node, which has length 6. Hence, we changed the value of `CERT_DEPTH` from 1 to 7. After 6, the vulnerability is the same as Fig. 4.10, shown as a dotted line here.

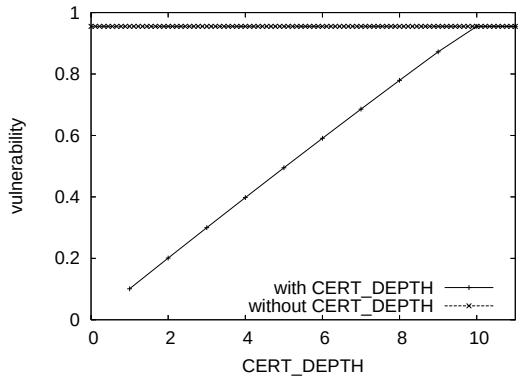


Figure 4.13: Effect of limit on chain length on vulnerability

As another example of acceptance criteria, we can use "path independence" proposed in [33]. This acceptance criteria requires k independent paths from *src* to *dst* for node *src* to be able to use the public key of *dst* in the certificate graph. To

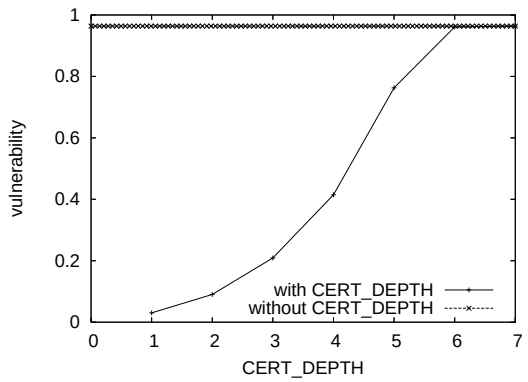


Figure 4.14: Effect of limit on chain length on vulnerability

find independent paths, the authors propose to use the min-cut size of a certificate graph from *src* to *dst*. Since *src* only uses the public key of *dst* if the min-cut size of a certificate graph from *src* to *dst* is at least k, the adversary needs to know at least k private keys.

In the current Internet, SSL/TLS [13] is one of the most commonly used protocols based on certificates. In SSL/TLS, most of the websites that have certificates signed by a CA, such as VeriSign, do not have alternate certificates signed by other CA. In other words, there is only one chain from one node to another. For this type of certificate graphs, path independence cannot be used.

The other commonly used protocol based on certificates is PGP [40]. PGP certificate graphs have the properties of small world [37]. The certificate graph in Fig. 4.15 is an example of small world graphs. Fig. 4.16 shows how vulnerability changes as k changes for this example certificate graph. As k increases, the vulnerability decreases.

In both examples of acceptance criteria, the graphs in Fig. 4.13 and Fig. 4.16 show that a stricter acceptance criteria reduces the vulnerability of certificate graphs. However, it also increases the number of valid public keys that cannot satisfy the stricter acceptance criteria. For example, in Fig. 4.13, the "avg usable keys" is the

average number of public keys a node can use. This increases as the depth limit increases. For the certificate graph in Fig. 4.15, the average number of public keys a node can use decreases from 5 to $\frac{14}{6} \sim 2.7$ as k increases from 2 to 3.

Also, according to the analysis in [8], the degrees of nodes in a self-organized certificate graph follow Zipf's distribution. In other words, most nodes in a self-organized certificate systems have a very small number of outgoing edges. In the Fig. 9 in [8], about half of the nodes in the largest strongly connected component of the 2001 PGP graph have fewer than three outgoing edges, and about 30% of the nodes have only one outgoing edge. Therefore, when the path independence is applied as the acceptance criteria, a large k may cause many public keys to become unusable by other nodes. In the previous example of the 2001 PGP graph, $k \geq 3$ will cause half of the nodes not to be able to use any public keys in certificate chains of length at least 2.

Clearly, there is a trade-off between the vulnerability of a certificate graph and the usability of the public keys in the certificate graph. Hence, acceptance criteria needs to be chosen and configured very carefully. This metric of vulnerability can help system administrators balance the resilience against impersonation attacks and the usability of the public keys in certificate graphs.

4.7 Vulnerability of Many Revealed Keys

As shown in the previous section, when an acceptance criteria requires more than one certificate chain from a node *src* to a node *dst* for node *src* to accept the public key in the certificate chain as the public key of *dst*, the vulnerability of a certificate graph can change depending on how many private keys are revealed to an adversary. Theorem 22 is modified here to take the case where many private keys are revealed to an adversary into the consideration.

Theorem 23 *Let G be a certificate graph and src and dst be any two distinct nodes*

in G. Let D be a set of nodes in G where the private key $r.d$ of each node d in D is revealed to an adversary. The adversary can impersonate node dst to node src if and only if $src \neq d$ for any node d in D and one of the following two conditions holds.

i. $d = dst$ for some node d in D and G has a set of certificate chains from src to dst that satisfies the acceptance criteria of G.

ii. There is no certificate (src, dst) and the set of certificate chains, in which each chain consists of a correct certificate chain from src to some node d in D that does not contain any certificate issued by node dst and a forged certificate (d, dst'), satisfies the acceptance criteria of G.

The vulnerability of the set D is defined as follows:

$$V(D) = \frac{|IMP(D)|}{(n - |D|) \times (n - 1)},$$

where $IMP(D) = \{(src, dst)|$ the adversary can impersonate dst to src using private keys of nodes in $D\}$ and n is the number of nodes in G. Let G be a certificate graph and there can be at most x private keys revealed to an adversary, then the vulnerability of graph G with x revealed keys, denoted $V(G, x)$, is defined as follows:

$$V(G, x) = \max_{D \subseteq G, |D| \leq x} V(D)$$

Note that this definition generalizes the definition of $V(G)$, which is equal to $V(G, 1)$.

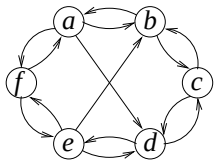


Figure 4.15: An example of a self-organized certificate graph

For the example certificate graph in Fig. 4.15, assume that the acceptance criteria of path independence with $k = 2$ is applied. Also, assume that the private keys of nodes b and d are revealed to an adversary. There is no certificate (a, c), and there are certificates (a, b) and (a, d), so the adversary can impersonate node c to node a. Also, there is no certificate (f, c), and there are certificate chains $(f, a)(a, b)$ and $(f, e)(e, d)$ that do not contain c, so the adversary can impersonate node c to node f. There are 16 node pairs (src, dst) such that the adversary can impersonate dst to src using the private keys of nodes b and d, so the vulnerability of $\{b, d\}$ is $\frac{16}{20}$. This is also the maximum vulnerability of the example certificate graph when $x = 2$, so $V(G, 2) = \frac{16}{20}$.

Fig. 4.16 shows how the vulnerability of the certificate graph in Fig. 4.15 changes as the number of revealed private keys changes. We applied the acceptance criteria of path independence with the parameter k from 1 to 3, and changed the number of revealed private keys x from 1 to 6. As the number of revealed private keys increases, the vulnerability increases. As long as the number of revealed private keys is less than k, the vulnerability is limited to explicit damage.

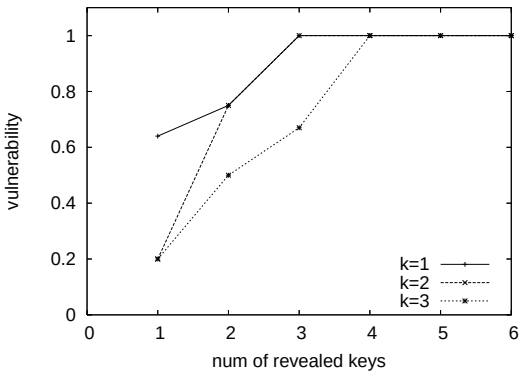


Figure 4.16: Vulnerability of many revealed keys

Chapter 5

Related Work

Several papers have investigated the use of certificates for confidentiality, authentication, and authorization. We summarize the results of these papers in the following paragraphs.

Architectures for issuing, storing, discovery, and validating certificates in networks are presented in [35, 7, 31, 18, 6, 12, 16, 19, 28]. In a large scale network such as today's Internet, one cannot expect to have a central authority to issue, store, and validate all the certificates. A distributed system, where each user participates in issuing, storing, and validating certificates is desirable in such a network.

In [39] and [25], distributed architectures for issuing certificates, particularly in mobile networks, are presented.

In [39], Zhou and Haas present an architecture for issuing certificates in an ad-hoc network. According to this architecture, the network has k servers. Each server has a different share of some private key rk. To generate a certificate, each server uses its own share of rk to sign the certificate. If no more than t servers have suffered from Byzantine failures, where $k \geq 3t + 1$, then the resulting certificate is correctly signed using the private key rk, thanks to threshold cryptography. The resulting certificate can be verified using the corresponding public key which is

known to every node in the ad-hoc network.

In [25], Kong, Perfos, Luo, Lu and Zhang presented another distributed architecture for issuing certificates. Instead of employing k servers in the ad-hoc network, no special nodes such as servers are in the network and every node in the network is provided with a different share of the private key rk. For a node u to issue a certificate, the node u forwards the certificate to its neighbors and each of them sign the certificate using its share of rk. If node u has at least $t+1$ correct neighbors (i.e. they have not suffered from any failures), then the resulting certificate is correctly signed using the private key rk.

In [28], Li, Winsborough, and Mitchell presented a role-based trust management language RT_0 and suggested the use of strongly typed distributed certificate storage to solve the problem of certificate chain discovery in distributed storage. However, they do not discuss how to efficiently assign certificates among the distributed storages. By contrast, our work focuses on minimizing storage overhead in certificate dispersal among the users while they have enough certificates so that there is no need for certificate chain discovery.

In [2], Ajmani, Clarke, Moh, and Richman presented a distributed certificate storage using peer-to-peer distributed hash table. This work assumes dedicated servers host a SDSI certificate directory and focuses on fast look-up service and load balancing among the servers. By contrast, our work assigns certificates to users such that there is no need for look-up and there are no dedicated certificate storage servers. Our work also focuses on efficient use of storages in all users in network.

Perhaps the closest work to the certificate dispersal is [22] where the authors, Hubaux, Buttyán, and Capkun, investigated how to disperse certificates in a certificate graph among the network nodes under two conditions. First, each node stores the same number of certificates. Second, with high probability, if two nodes meet

then they have enough certificates for each of them to obtain the public key of the other. By contrast, our work is based on two different conditions. First, different nodes may store different number of certificates, but the average number of certificates stored in nodes is minimized. Second, it is guaranteed (i.e. with probability 1) that if two nodes meet then they have enough certificates for each of them to obtain the public key of the other (if there exists a chain between them in the chain set).

Later, the same authors have showed in [9] that a lower bound on the number of certificates to be stored in a node is $\sqrt{n} - 1$ where n is the number of nodes in the system. Our work here shows that finding an optimal dispersal of a given chain set is NP-complete, and presents three polynomial-time algorithms which compute optimal dispersal of chain sets in three classes of practical interests and two extensions of these algorithms for more general classes of chain sets.

Zheng, Omura, Uchida, and Wada presented algorithms that compute optimal dispersals for strongly-connected graphs and directed graphs in [38]. The same authors also showed the tight upper bounds in these two classes of certificate graphs.

A public key infrastructure based on certificates is scalable and efficient in issuing and validating certificates but cannot tolerate Byzantine failures. In particular, if one node suffers from Byzantine failure, then this node can successfully impersonate any other node that is reachable from this node in the certificate graph of the network. This vulnerability to Byzantine failures is not unique to our certificate work. In Section 4, we have identified a metric to evaluate the damage from this type of attacks.

The metric of vulnerability can be used in any certificate system. For example, X.509 [1], SSL/TLS [13], PGP [40], and SDSI/SPKI [17, 35]. In any of these certificate systems, when a private key of some node is revealed to an adversary, the adversary may successfully impersonate nodes to other nodes in the system. In

other words, the certificate systems may be vulnerable to impersonation attacks.

Many researchers proposed mechanisms to evaluate certificate chains to mitigate this vulnerability. Tarah and Huitema [36] investigated using the path length as acceptance criteria. In [33], Reiter and Stubblebine investigated how to increase assurance on authentication with multiple independent certificate chains. They introduce two types of independent chains, disjoint paths (no edge is shared by any two chains) and k-connective paths (k certificates need to be compromised to disconnect all these paths). This paper shows that there are no polynomial-time algorithms for locating maximum sets of paths with these properties and presents approximation algorithms. Beth, Borcherding, and Klein [5] and Maurer [30] proposed an acceptance criteria based on probabilities. In PGP [40], users can limit the length of acceptable certificate chains and also require certain number of certificate chains to accept the public key of destination node. Levien and Aiken [27] presented an analytical model of different types of attacks and compared the resilience of acceptance criteria in [30] and [33] based on this model. The same authors also suggested another acceptance criteria based on the max flow algorithm. In [34], Reiter and Stubblebine suggested a number of guiding principles for the design of acceptance criteria.

Chapter 6

Conclusion

A certificate system is a useful public key infrastructure for distributed systems. A certificate can be stored anywhere in the system and can be used by any user who knows the public key of the issuer of the certificate. We have proposed a new way of distributing certificates that minimizes the communication overhead on the third party, called a certificate dispersal. Certificate dispersal assigns certificates to users in the system such that the two users that want to securely communicate with each other do not need to contact any third party for certificates. We showed that computing an optimal dispersal is NP-Complete when the dispersal cost is defined as an average number of certificates stored in each user, given a certificate chain set. We also presented several classes of certificate graphs and chain sets for which optimal dispersals can be computed in polynomial-time. Algorithms for these classes are also shown and proven to compute optimal dispersals. For a dynamic certificate system, we also devised a stabilizing dispersal protocol.

We have defined a metric called vulnerability that measures the potential scope of damage that an adversary with revealed private keys could incur to the system. We show that the vulnerability of a certificate graph is affected by the graph topology, dispersal, and acceptance criteria. One can use the vulnerability

measure as a design criteria of certificate systems, given the system requirements on dispersal cost and vulnerability.

As future work, we would like to build an application that utilizes the dispersal and vulnerability. For example, a large-scale distributed system where a relatively small number of autonomous systems are cooperating could benefit from dispersal and vulnerability. Between the coordinators of autonomous systems, one can expect that the certificate system would not change rapidly. We can compute an optimal dispersal between coordinators periodically, or run the dynamic dispersal protocol. At the same time, the vulnerability metric could be a guideline in whether to issue certain certificates or not, or even in deciding the price of issuance for those certificates.

Bibliography

[1] C. Adams, S. Farrell, T. Kause, and T. Mononen. Internet X.509 public key infrastructure – certificate management protocol (CMP). RFC 2510, 1999.

[2] S. Ajmani, D. E. Clarke, C.-H. Moh, and S. Richman. ConChord: Cooperative SDSI certificate storage and name resolution. In *LNCS 2429 Peer-to-Peer Systems: First International Workshop, IPTPS 2002*, pages 141–154, March 2002.

[3] A. Arora and M. G. Gouda. Distributed reset. In *Proceedings of the 22nd International Conference on Fault-Tolerant Computing Systems*, 1990.

[4] A. Arora and M. G. Gouda. Closure and convergence: A foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering*, 19:1015–1027, November 1993.

[5] T. Beth, M. Borcherding, and B. Klein. Valuation of trust in open networks. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS '94) LNCS 875*, pages 3–18. Springer-Verlag, 1994.

[6] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote trust-management system version 2. RFC 2704, 1999.

[7] S. Boeyen, T. Howes, and P. Richard. Internet X.509 public key infrastructure operational protocols - LDAPv2. RFC 2559, 1999.

[8] S. Capkun, L. Buttyán, and J.-P. Hubaux. Small worlds in security systems: an analysis of the PGP certificate graph. In *Proceedings of the ACM New Security Paradigms Workshop*. ACM Press, 2002.

[9] S. Capkun, L. Buttyán, and J.-P. Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(1):52–64, 2003.

[10] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *OSDI: Symposium on Operating Systems Design and Implementation*, 1999.

[11] N.-S. Chen, H.-P. Yu, and S.-T. Huang. A self-stabilizing algorithm for constructing spanning trees. *Inf. Process. Lett.*, 39(3):147–151, 1991.

[12] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[13] T. Dierks and E. Rescorla. The TLS protocol version 1.1. Internet Draft (draft-ietf-tls-rfc2246-bis-08.txt), 2004.

[14] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[15] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*. ACM, 1990.

[16] Y. Elley, A. Anderson, S. Hanna, S. Mullan, R. Perlman, and S. Proctor. Building certificate paths: Forward vs. reverse. In *Proceedings of the 2001 Network and Distributed System Security Symposium (NDSS '01)*, pages 153–160, February 2001.

[17] C. Ellison. SPKI requirements. RFC 2692, 1999.

[18] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. RFC 2693, 1999.

[19] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. dRBAC: distributed role-based access control for dynamic coalition environments. In *Proceedings of 22nd International Conference on Distributed Computing Systems (ICDCS '02)*, pages 411–420, 2002.

[20] M. G. Gouda and E. E. Jung. Stabilizaing certificate dispersal. In *2005 Symposium on Self Stabilizing Systems - LNCS 3764*. Springer-Verlag, 2005.

[21] M. G. Gouda and N. Multari. Stabilizing communication protocols. *IEEE Transactions on Computers, Special Issue on Protocol Engineering*, 40(4):448–458, April 1991.

[22] J.-P. Hubaux, L. Buttyán, and S. Capkun. The quest for security in mobile ad hoc networks. In *Proceedings of the 2001 ACM International Symposium on Mobile ad hoc networking & computing*, pages 146–155. ACM Press, 2001.

[23] E. Jung, E. S. Elmallah, and M. G. Gouda. Optimal dispersal of certificate chains. In *Proceedings of the 18th International Symposium on Distributed Computing (DISC '04)*. Springer-Verlag, 2004.

[24] R. Kohlas and U. Maurer. Confidence valuation in a public-key infrastructure based on uncertain evidence. In *Proceedings of PKC 2000, LNCS 1751*. Springer-Verlag, 2000.

[25] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing robust and ubiquitous security support for wireless mobile networks. In *Proceedings of Ninth International Conference on Network Protocols (ICNP '01)*, pages 251–260, 2001.

[26] B. Krebs. The new face of phishing. WashingtonPost.com, 2006.

[27] R. Levin and A. Aiken. Attack resistant trust metrics for public key certifications. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[28] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, 2003.

[29] D. Malkhi and M. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.

[30] U. Maurer. Modeling a public-key infrastructure. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS '96)*. Springer-Verlag, 1996.

[31] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet public key infrastructure online certificate status protocol - OCSP. RFC 2560, 1999.

[32] S. P. Nielsen, F. Dahm, M. Lüscher, H. Yamamoto, F. Collins, B. Denholm, S. Kumar, and J. Softley. Lotus notes and domino r5.0 security infrastructure revealed, 1999.

[33] M. K. Reiter and S. G. Stubblebine. Resilient authentication using path independence. *IEEE Transactions on Computers*, 47(12):1351–1362, December 1998.

[34] M. K. Reiter and S. G. Stubblebine. Authentication metric analysis and design. *ACM Transactions on Information and System Security (TISSEC)*, 2(2):138–158, 1999.

[35] R. L. Rivest and B. Lampson. SDSI – A simple distributed security infrastructure. Presented at CRYPTO '96 Rumpsession, 1996.

[36] A. Tarah and C. Huitema. Associating metrics to certification paths. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS '92) LNCS 648*. Springer-Verlag, 1992.

[37] D. Watts. *Small Worlds*. Princeton University Press, 1999.

[38] H. Zheng, S. Omura, J. Uchida, and K. Wada. An optimal certificate dispersal algorithm for mobile ad hoc networks. In *Proceedings of Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar'04)*, 2004.

[39] L. Zhou and Z. J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, 1999.

[40] P. Zimmerman. *The Official PGP User's Guide*. MIT Press, 1995.

Vita

Eunjin Jung was born in Seoul, Korea on November 26, 1976, the daughter of Byungkun Chung and Hyesook Koo. She attended Seoul Science High School from 1992 to 1994. She was a Class Valedictorian when she received the degree of Bachelor of Science from Seoul National University in February 1999. She entered the Graduated School of The University of Texas at Austin in August 1999 and received the degree of Master of Science in Computer Sciences in May 2002. She received Outstanding Teaching Assistant Excellence Award from the Department of Computer Sciences in December 2005 and William S. Livingston Outstanding Graduate Student Employee Award as Teaching Assistant from The University of Texas at Austin in May 2006. Her research interests are security, and fault-tolerance in computer networks and distributed systems, and Digital Rights Management. She has published nine refereed journal and conference papers on topics in security in networks and distributed systems.

Permanent Address: 2501 Lake Austin Blvd. Apt C202
Austin, TX 78703
U.S.A.

This dissertation was typeset with LaTeX 2$_\varepsilon$[1] by the author.

[1] LaTeX 2$_\varepsilon$ is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab.