ABSTRACT

QINGLIN JIANG Improving the Robustness of Webs of Trust. (Under the direction of Dr. Douglas S. Reeves and Dr. Peng Ning). The correct recognition of a user's public key is very important for many security functions, such as confidentiality, integrity and non-repudiation. If we mistakenly recognize an illegitimate public key as legitimate, then these security functions may be compromised. In distributed webs of trust systems, each user's public-key information is provided by other users. Because users can be unreliable (untrustworthy, malicious, compromised users or who make mistakes), the correctness of the public-key information they provided remains a question. For this reason, a method to verify the correctness of the user-provided public-key information is very much needed.

Previous works have suggested the use of redundancy to compute the trustworthiness on user-provided public key information. However, the problem of how to improve the trustworthiness has never been considered. In this paper, we will focus on the problem of how to improve the trustworthiness of user-provided public-key information. First, we observe that the trustworthiness computed on a public key may be inaccurate if users claim multiple false identities and/or (either legitimately or illegitimately) possess multiple public keys. We explain and show that the result of trust computation can be made more accurate if we also consider identities. Second, we analyze *conflicting* certificates and show that it can be used to detect malicious users and improve the trustworthiness on public keys. Third, we show that the current webs of trust system, i.e. PGP, is not robust in the presence of unreliable users. Its robustness can be significantly improved by the two kinds of certificate *recommendation* methods we have proposed.

The first method can be used to improve the robustness of the whole webs of trust system to any desired degree by issuing a minimal set of additional certificates. These recommendations are also made very user-friendly by taking into consideration user's preference and non-compliance. The second recommendation method works differently. It is based on probability theory and can be used to increase the robustness of any single public key as well as the entire webs of trust system. It can guarantee the correctness of a user's public key by over 99.99% probability with only a moderate number of additional certificates; even in the presence of a large number of unreliable users. The applications of both recommendation methods will result in richly-connected and very robust webs of trust systems. In the last recommendation, we present a very efficient and robust mechanism to apply the webs of trust system in wireless ad-hoc networks. The specific problem is how to distribute public key certificates to each user such that users can authenticate each other. Our mechanism enables users to exchange certificate path information so they can easily find certificate paths and authenticate each other creating greater efficiency and requiring less communication overhead. This mechanism is also very robust because it considers the case of network partitions.

In addition, our mechanism provides more robust public-key authentication as it can construct and find multiple certificate paths between users. For all the works presented in this paper, we illustrate their concepts and show the results on practical web of trust PGP keyrings.

Improving the Robustness of Webs of Trust

by

Qinglin Jiang

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Doctor of Philosophy

Computer Engineering

Raleigh

2006

Approved By:

To my wife Xiaoping Xie
and my parents Lindong Jiang, Shuman Zhao
for their love and support

Biography

Qinglin Jiang was born in Liaoning, China. He received his Bachelor degree in Aerospace Engine from Beijing University of Astronautics and Aeronautics in China in 1996 and a Masters degree in Computer Application in Beijing Polytechnic University in China in 1999. Upon graduation, he worked for Lenovo Inc. and elong.com Inc. in Beijing, China. He was admitted to North Carolina State University, Department of Electrical and Computer Engineering, in the Fall of 2000 where he studied computer engineering as a Ph.D student under the supervision of Dr. Douglas S. Reeves and Dr. Peng Ning. His research interests include distributed public-key infrastructures and mobile ad-hoc networks.

Acknowledgements

First of all, I would like to express my great appreciation to my Ph.D. advisors, Dr. Douglas S. Reeves and Dr. Peng Ning. My research work and this dissertation would not be possible without their guidance, encouragement and support. Dr. Reeves and Dr. Ning have led me through my entire Ph.D research. They always looked much further into my research direction and gave great feedback for my work. I would also like to thank Dr. Gregory T. Byrd, and Dr. Wenye Wang for providing comments on my dissertation and serving on my advisory committee. Thanks also to Dr. Matt Stallman for his valuable discussions and suggestions on various papers. Many thanks go to past and current members of the Cyber Defense Laboratory, Kehang Wu, Fang Feng, Qinghua Zhang, Pan Wang, Dingbang Xu, and others. Ive enjoyed working with them very much.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background Information

Cryptography plays a central role in today's information security world. There are two kinds of cryptography techniques: symmetric-key techniques and public-key techniques. In symmetric-key techniques, two communication users share a common secret key. To provide confidentiality, a message is encrypted using the secret key and can only be decrypted by using the same key. For integrity checking, a MAC (Message Authentication Code) is generated using the secret key and is sent with the message which can only be verified by using the same secret key. In public-key techniques, each user owns a public key and the corresponding private key. The private key is usually kept confidential by its owner (or by a trusted third party), but the public key is supposed to be known by all communication users. It works in a different way from symmetric-key techniques. For confidentiality, a message is encrypted using the receiver's public key, and can only be decrypted by using the receiver's private key. For integrity and non-repudiation, a digital signature is generated for the message using the sender's private key and can only be verified using the sender's public key.

Symmetric key techniques have long been used for communication security. It is used in many aspects of the cryptography, such as key distribution, encryption, and authentication. The birth of public-key techniques brings us many nice features[48] that we may take advantage of. First, for confidentiality purposes, symmetric-key technique requires the setup of pairwise keys which means any pair of communication users need to share a unique secret key. While in public-key technique, each user will need only one public key. And instead of requiring secrecy, the public key is supposed to be publicly known by all users. Second, the symmetric-key technique usually requires a trusted on-line server to distribute secret keys through a secure channel. While in public-key technique, it can be replaced by an off-line trusted server plus any means of distributing public keys. Third, the notable security features provided by public-key technique, such as non-repudiation and data origin authentication, are very hard to be implemented cost-effectively by symmetric techniques. However, the many features of public-key technique has its own expenses. The computation time of public-key operation is usually more expensive than that of symmetric-key. In consideration of this issue, a common technique used in the security world is to use public-key technique to setup the secret key and then use the secret key for secure communication sessions.

Public-key technique is often recognized as an ideal solution for key management [48]. There are several techniques for distributing public keys[48], such as point-to-point delivery, trusted public-key registry, on-line trusted server, off-line server with certificates, identity-based systems and implicit-certified public keys. Point-to-point delivery means users exchange public keys directly. The exchange is usually done over a trusted channel. For example, we can write the public key on a piece of paper and exchange it. Point-to-point delivery is often used in bootstrapping trust for other public-key distribution methods. Trusted public-key registry and on-line trusted server provide a central repository where users may retrieve public keys directly. In retrieving the public keys from the central repository, there is no need of confidentiality protection. But integrity is required, because we have to make sure that the public key is not tampered during the transmission. In identity-based and implicit-certified public-key systems, a user's public key can be reconstructed from publicly available information such as name and address. The corresponding private key can be generated by the user itself or by a trusted third party T using both publicly available information and T's private key.

In addition to the methods we discussed above, perhaps one of the most commonly used technique to distribute public keys is public-key certificates. Examples are PKI(Public-key Infrastructure), such as X.509 [60] and PGP [65]. In these systems, the public-key certificate, which contains a signed name-to-key binding, is used to distribute public keys. Throughout this paper, we will simply refer "public-key certificate" as "certificate". Certificates are usually issued by a CA(certificate authority). In practice, X.509 consists a hierarchical structure of CAs with a root CA and is often referred as a "hierarchical trust" system. It is popularly used in today's information security infrastructures, such as Verisign[2]. In addition, it appears that "webs of trust" systems apply when "hierarchical trust" systems are not available or too expensive. A typical example is the freely available PGP[65] which is very popular among personal computer users. In the "webs of trust" system, each user becomes a certificate authority and issues certificates between each other. Users of webs of trust systems are connected to each other by the certificates they issued. They authenticate each other's public key using the certificate chains that connect them together. The webs of trust system is very different from the hierarchical trust system and has its own advantages. One of the obvious advantages is that webs of trust system is completely composed of and controlled by individual users. It doesn't have any centralized controls and neither is controlled by any company or government. The best thing about PGP is that it is free. You don't have to pay a single penny to get your public key signed and get issued a public-key certificate. By contrast, hierarchical trust systems are usually controlled by some companies or the government. The process to get a certificate issued by them is usually very complicated and involves a number of documents, verifications, and legal issues. And most of these certificates won't come free.

For these reasons, it is not surprising to see why webs of trust systems have become popular in the webs of users. We believe that it is a valuable topic to work on and can benefit a large amount of users. In the following section, we will explain some details of how webs of trust systems work and the research problems that we'll be working on in this paper.

1.2 The problem of public-key authentication in webs of trust system

The main content of webs of trust systems are certificates. Certificates are generated in the following way. The first step is key generation. A user will normally generate a pair of public and private key for themselves. They are generated using public key algorithms, such as RSA[56], DSS[3], and others. The private key will be kept secret by the owner while the public key is made widely available to other users. One of the properties that make public key and private key very useful is that anything encrypted by the public key can only be decrypted by the private key. If someone encrypts a message with Alice's public key, only Alice will be able to decrypt it. The other property is that anything signed by the private key can only be verified by the public key. If Alice applies the decryption function (usually called signing function) to a message, the only way to verify the integrity of the message is to use Alice's public key. By using the pair of public and private keys, we are able to achieve the major objectives of security, which include confidentiality, integrity, data-origin authentication, and non-repudiation. For all these security functions to work, the very basic requirement is that Alice's public key needs to be distributed to and recognized by other users correctly. The way to distribute Alice's public key in webs of trust systems is to use public-key certificate. A public-key certificate is a signed name-to-key binding. The name-to-key binding is a binding of the user's name and public key, such as Alice and k_1. For Bob to issue Alice a certificate, Bob will generate a digital signature over Alice's name-to-key binding using his private key and associate the digital signature with it. While the digital signature in the certificate guarantees that the name-to-key binding will not be tampered by others, it doesn't guarantee the correctness of it. The reason is that Bob has the sole control of the content of the certificate is because the certificate is generated by them. For example, suppose Bob is malicious: He can generate a key pair of public key k and private key d. He then issued a certificate which certify a name-to-key binding of Alice and k. In this situation, any confidential information sent to Alice that is encrypted with k would be readable by Bob, which is undesirable. The ability to authenticate Alice is also compromised because Bob could generate digital signatures which would make other users believe that the information originates from Alice. For this reason, we must make sure that we recognize a user's public key correctly.

In a hierarchical public-key infrastructure system, such as X.509 [60]. Public-key certificates are issued by Certificate Authority (CA). We usually assume that the public-key certificates all contain correct name-to-key bindings because all CAs are assumed to be secured and trusted. In a non-hierarchical public-key infrastructure system, such as PGP webs of trust systems[65], each user becomes a certificate authority. In webs of trust systems, it may be risky to expect all certificates to contain correct name-to-key bindings, because not all users are fully secured and trusted. Some users may be malicious and issue public-key certificates which contain incorrect name-to-key bindings. Other users may be unreliable, they may issue certificates which contain errors in the name-to-key bindings. And some user's computer systems may be compromised and be used by malicious users to issue public-key certificates which contain incorrect name-to-key bindings. For these reasons, a

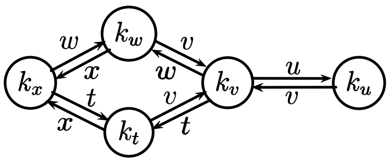


Figure 1.1: A sample certificate graph

method that can be used to verify the correctness (authenticity) of a name-to-key binding in webs of trust systems is very much needed.

We have explained the problem of public-key authentication in webs of trust systems. In the next section, we will introduce the certificate graph model used in the paper.

1.3 The certificate graph

In this section, we will introduce the basic notation and model used in webs of trust systems. There are extended models based on different assumptions that will be explained in the corresponding chapters and sections.

First, we briefly introduce some notions in webs of trust systems. The webs of trust system is composed of many entities, such as users and computers. An entity may be represented by a *name*, such as "Bob" and "Alice". A *public key certificate* is a signed name-to-key binding represented by a triple $\langle x, k_x, s(k_y)\rangle$, where x is a name, k_x is a public key, and $s(k_y)$ is a digital signature (generated using the private key corresponding to k_y) over the binding of x and k_x. A certificate chain is a sequence of certificates where the public key of one certificate can be used to verify the digital signature associated with the previous certificate of the chain.

A set of certificates can be represented by a certificate graph. A certificate graph G is a simple directed graph and consists of a set V of vertexes and a set E of edges. In this graph, a vertex labeled k in V represents the public key k. There is an edge labeled x from vertex k_y to vertex k_x if and only if there exists a certificate $\langle x, k_x, s(k_y)\rangle$. That is, the key k_y is used to sign a certificate binding x to k_x. A certificate chain is represented by a directed path in the certificate graph. Figure 1.1 depicts a sample certificate graph. In this graph, there are 5 vertexes and 10 edges that represent 5 public keys and 10 certificates, respectively. The edge from k_x to k_w, labeled with w represents the certificate $\langle w, k_w, s(k_x)\rangle$.

We have explained the basic certificate graph model used in this paper. In the next section, we will briefly introduce our research works and contributions in this paper.

1.4 Works and Contributions

Our research works in this paper include the following four parts. The first one is an improved model on computing the trustworthiness of user's public keys. The second and the third works present two kinds of certificate recommendation methods to increase the trustworthiness of public keys by guiding users to issue additional certificates. In the last work, we will talk about issues of public-key authentication in mobile ad-hoc networks. They are briefly introduced as follows.

1.4.1 Computing webs of trust.

This work focuses on trust computation on user-issued name-to-key bindings. That is, computing the assurance on the correctness of user-issued name-to-key bindings. The assurance represents the level of resistance against unreliable users, such as the total number of malicious users. Previous works have observed that multiple certificate chains can be used to compute the assurance, but haven't addressed the following issue. That is, a single malicious user may (either legitimately or illegitimately) possess multiple public keys, or (falsely) claim multiple identities, and therefore create multiple certificate chains to produce misleading trustworthiness on user's public keys. Our solution to this problem is to also consider identities in trust computation and use identity independent certificate chains as the measurement of trustworthiness of user's public keys. We prove the problem to find the number of identity independent certificate chains to be NP-Complete, and we present heuristics to compute it. We also investigate the implications of *conflicting certificates* (i.e., certificates which disagree about the public key of a user). We present several rules of detecting malicious users and show that the trustworthiness of user's public keys can be significantly improved by doing this.

1.4.2 Improving Webs of Trust.

Improving trust computation is to improve the assurance of the correctness of name-to-key bindings. Our motivation comes from the observation that in real-world webs of trust systems, i.e., PGP keyrings, users don't have rich connections between each other in the certificate graph. Thus they don't provide robust protection against malicious attacks even in the existence of a small number of malicious users. All existing works on trust computation mainly focus on the problem of how to compute the assurance of name-to-key bindings but have never addressed how to improve it. In this work, we propose a method for generating certificate *recommendations*. These recommendations guide the users in creating webs of trust that are richly connected and highly robust to attacks. The problem is modeled as a standard graph theory problem, i.e., the graph connectivity augmentation problem. We propose a heuristic method of graph augmentation for the certificate graph, and show experimentally that it is close to optimal. By using the heuristic, we can generate certificate recommendations which will increase the robustness of the entire webs of trust system to any desired level with a minimal set of additional certificates. We also investigate the impact of user preferences and non-compliance with these recommendations, and demonstrate that our method helps identify malicious users.

1.4.3 Constructing webs of trust.

The webs of trust system is constructed by the certificates that users issue to each other. These certificates are usually created at user's will without any directions. In this work, we present a new way to direct users to issue certificates to each other. Basically, we make very efficient certificate recommendations which direct users on how certificates should be issued to each other. The proposed method is based on probability theory. It can guarantee the correctness of user's public keys with over 99.99% probability by issuing only a moderate number of additional certificates in the presence of a large number of unreliable users. In addition, we also consider the case of user's non-compliance and show how our method will still work by making redundant recommendations. Moreover, we present an algorithm of detecting malicious users by using the recommendation method and show experimentally that it is very useful and efficient.

1.4.4 Webs of trust in MANETs.

Mobile ad-hoc networks(MANET) have caught a great deal of attention in the information world recently. The feature of not requiring infrastructure support makes ad-hoc networks an ideal solution for military tactical operations, emergency, and rescue missions, where the infrastructure support is not available. For example, in a mission of rescuing hostages, a marine troop of soldiers equipped with digital warfare systems may communicate with each other via the ad-hoc networks constructed by themselves without any extra devices. The problem we consider in this work is the application of webs of trust systems in MANET. In a network with infrastructure support, such as nowadays a wired network, all certificates may be stored in a on-line certificate directory where each user would be able to retrieve certificates whenever needed. But in the ad-hoc networks, due to the many limitations of mobile nodes, such as frequent disconnection, limited memory, power and bandwidth, it is impractical for any single node to serve as a central certificate directory. Previous works have suggested how to efficiently distribute public-key certificates to each user such that they can authenticate each other's public keys. But they haven't addressed the issues of network partitions and multiple certificate chains. We present a mechanism that enables users to exchange certificate path information so they can easily find certificate paths and authenticate each other. In different scenarios of network partitions, our mechanism will still work since it doesn't have a single point of failure. In addition, our mechanism provides more robust public-key authentication so it can construct and find multiple certificate chains between users.

1.4.5 Contributions

The contribution of this paper is as follows. First, we point out a very important issue in computing webs of trust. That is, a user may possess multiple public keys or multiple false ids in order to mislead us to believe bogus public keys. We address this issue by also considering identities in computing the assurance. Second, we discover that certificate conflicts can be used to increase the assurance. Third, we present two efficient certificate recommendation techniques to make more robust webs of trust systems. One recommendation technique is used to increase the robustness of an existing webs of trust system to any desired level. The other recommendation technique is used to direct users to construct a very robust webs of trust system with only a moderate number of certificates. Fourth, we present methods to detect malicious users and show experimentally that it is very effective. In the last, we present a more robust and efficient solution to authenticate public keys in the mobile ad-hoc networks.

1.5 Outline of this paper

The paper is organized as follows. Chapter 1 first introduces some background information for public key techniques and explains the security issues in computing trust in distributed systems. It also briefly explained the works presented in this paper, the contributions and the outline of this paper. Chapter 2 summarized related works and discuss some representative works in details. Chapter 3 presents a new model on computing webs of trust and shows how certificate conflicts may be used to improve the robustness of webs of trust. Chapter 4 presents a method of certificate recommendations which guides users to issue certificates and improve the assurance of public-key authentication. Chapter 5 presents a very efficient way to construct a robust webs of trust system. It allows us to construct a very robust webs of trust system with a moderate number of certificates. Chapter 6 discusses the problem of public-key authentication in mobile ad-hoc networks. It presents new challenges of webs of trust in mobile ad-hoc environments and presents solutions that are useful. Chapter 7 concludes this work and discusses future works.

Chapter 2

Related Works

In this chapter, we will introduce research works that are related to public-key authentication in the webs of trust system. We will briefly summarize some related works and choose some representative works to be discussed in detail.

2.1 Overview

X.509 [60] is the industry standard for public-key infrastructure and has been widely used in both enterprise and Internet applications. In this public-key infrastructure, CAs(Certificate Authorities) are used to issue public-key certificates. The CAs usually appear in hierarchical levels where the higher level CAs issue public-key certificates to those at the level below. The top most CA is usually called "root CA" and the bottom level CAs issue public-key certificates directly to users. Contrary to the hierarchical public-key infrastructure, PGP [65] represents the public-key infrastructure in webs of trust. In the webs of trust, there usually is no CA and any user can issue public-key certificates to any other user. As users issue certificates to each other, they form a web of trust relationship. There are also other public key infrastructures, such as SPKI/SDSI [22], Delegation Networks [8] and PolicyMaker [13]. They mainly focus on access control issues. They differ from X.509 and PGP in that they bind access control policies directly to public keys, instead of to identities. [14] discusses the possible threats for current public key infrastructures.

Existing works on computing the trustworthiness of webs of trust can be classified into two categories. In the first category, *partial trust* is used to compute the authenticity of a target public key. In [59] and [47], this authenticity is based on the trust value in a single certificate chain. Multiple certificate chains are used in [12] and [45] to boost the assurance of authenticity. In [52], insurance, which may be viewed as a means of reducing risk, it is used to calculate the authenticity of public keys. In [42], a maximum flow approach is presented to compute the authenticity on users' public keys.

In the second category of methods, there is no partial trust; a key is either fully authenticated, or else it is not authenticated. In this category, an upper bound is set on the number of participants that may be malicious. The representative method in this category is [53]. This method suggests using multiple public key-independent certificate chains to certify the same key. The authors show that if at most n public keys are compromised, a public key is probably correct if there are $n+1$ or more public key-independent certificate chains certifying it.

All methods for distributed trust computation assume there is some initial trust between selected users. Without such initial trust, there is no basis for any users to authenticate any public keys. In [20], it is shown that forging multiple identities is always possible in a decentralized system. We assume that the initial trust must be negotiated in an *out-of-band* way (such as by direct connection, or communication with a trusted third party), and that proof of identity is available during this initial phase.

Mobile ad-hoc networks [51] have caught a great deal of attention in recent years. For hierarchical public-key infrastrcutures, an important issue is that a single CA may be a vulnerable point in ad-hoc networks. If it is compromised, then the authenticity of all public keys would be compromised. Several research works [19, 57] have addressed this issue by using multiple nodes to collaboratively perform the function of a CA using threshold cryptosystems. In these works, [63] takes advantage of threshold secret sharing and proactive secret share update techniques to distribute the services of the CA to a set of specialized nodes. Each of these nodes can generate a partial signature and a combination of k such signatures is required to create a valid certificate. [44] uses a (k, n) threshold scheme to distribute an RSA certificate signing key to all nodes in the network. A nice feature of this work is that the system does not expose to any single point of compromise, single point of denial of service attack, or single point of failure. These works make it possible for a CA to securely generate certificates in ad-hoc networks.

For the "webs of trust" model, [33] is the first to present a method for public-key authentication in ad-hoc network. In this work, each user will collect a set of certificates from others and store them in its local certificate repository. To authenticate user y's public key, user x will merge her own and the certificate repository sent from y and tries to find a certificate chain from herself to y . The authors also present an improved version of their method in [15]. In this work, the basic approach is similar to [33] but it uses a different mechanism for each user to collect certificates. For hierarchical public-key infrastructures, a method of discovering certificate path in ad-hoc networks is presented in [32]. It discovers certificate paths by encoding the path information into the certificates themselves. The application of webs of trust system in MANET also requires a mapping of the node's name and its network address. There are schemes available to map a node's name to its ip address in the MANET, such as [7].

For security in wireless sensor networks, the main research focuses on pair-wise key establishment, such as the random shared key distribution scheme in [23]. Many papers [43, 21, 64] have later improved this method. Security problems in sensor networks are different from those in general ad-hoc networks due to their different natures. The main concern in sensor networks is due to node's limited computation, communication power, and memory constraints. While in ad-hoc networks, we are more concerned with its dynamic nature. Though the key-distribution mechanism in sensor network may apply in ad-hoc network, it may not be suitable or the best technique. For example, the limited computation power prohibits the application of public-key cryptography in sensor network, but we are able to take advantage of it in ad-hoc networks.

We have summarized some related works to trust computation. In the next sections, we will choose some representative works and discuss them in detail.

2.2 Pretty Good Privacy(PGP)

The real world webs of trust system is PGP[65]. In PGP, users issue public-key certificates to each other. There are three levels of validity, which are: valid, marginally valid, and invalid. Validity is the confidence the user has that a public key belongs to its claimed owner. The validity is computed by the PGP model. There are also three levels of trust that a user can place on a public key. These levels are: complete trust, marginal trust, and no trust. The trust of a public key means the user's belief regarding the validity of another public key signed by it. The level of trust on a public key is determined by each user. Note that in the context of PGP, the validity, instead of the level of trust, is the research problem that we will work on in this paper. To determine a valid public key, we will need either one trusted signature or two marginally trusted signatures. That is, it is either signed by one trusted public key or two marginally trusted public keys. The idea of using two marginally trusted signatures is because the odds that two users make the same mistake is probably small. The number of marginally trusted signatures required to determine a valid public key can also be set by the users. For example, Alice has two certificates. One is signed by public key k_{Mike} and issued to Bob's public key k_{Bob}. The other is signed by public key k_{Cindy} and also issued to Bob's public key k_{Bob}. If both public keys k_{Mike} and k_{Cindy} are marginally trusted by Alice, then PGP will determine for Alice that Bob's public key k_{Bob} is a valid public key. From above, we can see that PGP's trust model is very straight-forward and useful. Users just need to specify which public keys to trust and the remaining jobs will be all done by the PGP model. The results are very meaningful because it directly tells you if a public key is valid or not. One issue with PGP is that its ability to determine a valid public key is limited by the number of trusted and marginally trusted public keys. It may or may not be a good thing depending on the need of different users. If a user only cares about the validity of the public keys within a small group, then PGP is usually sufficient to do the job. However, if the user will be required to determine the validity of a large number of public keys, then their ability to do it will be limited by the number of trusted and marginally trusted public keys.

2.3 Valuation of trust in open networks

In Beth, Borcherding and Klein's model [12], the authenticity of a public key is computed as follows: The input is a direct graph with vertexes as entities and edges as trust relationships. There are two types of edges in this graph, the direct edge and the recommendation edge. A direct edge $A \rightarrow B$ represents A can authenticate B's public key. Associated with this edge is a value which ranges from 0 to 1 and it represents how much A believe that it has the correct public key for B. The recommendation edge $A \rightsquigarrow B$ represents A trusts B to authenticate other entities or recommend further. Associated with this edge is a value which ranges from 0 to 1 and it represents how much A recommends B to authenticate other entities or recommend further. Given that A wants to authenticate B's public key, A will take as input all paths from A to B whose last edge is a direct edge and whose other edges are recommendation edges; computing a value which ranges from 0

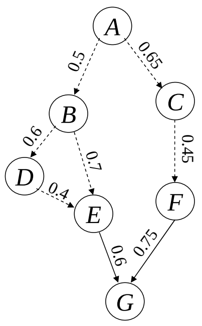


Figure 2.1: Computing combined trust value

to 1. The value represents how much A should believe that it has the correct public key for B. The actual computation rules are as follows.

1. If there is path $A \rightsquigarrow \cdots \rightsquigarrow C$ with recommendation value α_1 and a recommendation edge $C \rightsquigarrow D$ with recommendation value α_2, then the recommendation value for the path $A \rightsquigarrow \cdots \rightsquigarrow C \rightsquigarrow D$ is $\alpha_1 \cdot \alpha_2$.

2. If there is path $A \rightsquigarrow \cdots \rightsquigarrow C$ with recommendation value α_1 and a direct edge $C \rightarrow D$ with trust value α_2, then the trust value for the path $A \rightsquigarrow \cdots \rightsquigarrow C \rightarrow D$ is $1 - (1 - \alpha_2)^{\alpha_1}$.

3. If there are m different direct edges $C_i \rightarrow D(1 \leq i \leq m)$ and for each i, there are n_i distinct paths from A to D ending at $C_i \rightarrow D$ with trust values $\alpha_{i,1}, \ldots, \alpha_{i,n_i}$, then the combined trust value is

$$\alpha_{com}(A, D) = 1 - \prod_{i=1}^{m} \sqrt[n_i]{\prod_{j=1}^{n^i} (1 - \alpha_{i,j})}$$

The example in figure 2.1 shows how this actually works.

By rule 1 and 2, the three paths $A \rightsquigarrow B \rightsquigarrow D \rightsquigarrow E \rightarrow G$, $A \rightsquigarrow B \rightsquigarrow E \rightarrow G$, and $A \rightsquigarrow C \rightsquigarrow F \rightarrow G$ have trust values of 0.1, 0.37 and 0.33, respectively. And by rule 3, the combined trust value is $\alpha_{com}(A, D) = 0.5$.

This work is based on probabilistic model and the computed trust value depends on the trust values and recommendation values in the trust profiles established for each entity. It makes it possible to differentiate these public keys based on the computed trust values. If we set a threshold for the computed trust value, then we may determine that any public key with a combined trust value higher than the threshold to be a correct public key. On the contrary, the public keys with combined trust values lower than the threshold may not be determined to be correct. Of course the threshold can be tuned to make the method more accurate. But generally, this kind of method will always produce false positives and false negatives. That is, it is always possible that a public key whose combined trust value is lower than the threshold turns out to be a correct one, and a public key whose combined trust value is higher than the threshold turns out to be an incorrect one. Also the combined trust value will also be affected by the trust and recommendation values used in the computation. If these values are not set correctly, then the result of computation may be misleading. Also as shown in [54], this model is subject to malicious attacks. In figure 2.1, if C is a malicious user and creates some bogus vertexes and connects itself to G through these bogus vertexes, then the combined trust value will be manipulated. This work also provides the idea of constrained set for its trust computation. That is, it can exclude certain nodes in its trust computation. However, this requires the user to be familiar with the users used in its trust computation. Else, the user won't have any idea on whom to exclude.

2.4 Modeling a public-key infrastructure

Similar to [12], Maurer[45]'s model also takes as input the directed certificate graph. A vertex in the graph represents an entity and an edge represents a statement. There are four types of statements in this model as follows.

1. *Authenticity of public keys.* It represents an entity's belief that a specific public key belongs to another entity.
2. *Trust.* It represents an entity is trustworthy in issuing certificates or recommendations.
3. *Certificate.* It represents a certificate for an entity's public key issued and signed by another entity.
4. *Recommendations.* It represents a recommendation for an entity issued and signed by another entity.

All these statements can appear in the form of certificates and are represented by an edge in the certificate graph. The semantics of these edges are subtly different from those in [12]. A direct edge $X \rightarrow Y$ denotes the fact that the entity holds a certificate for Y's public key(allegedly) issued and signed by entity X. Similarly, a recommendation edge $X \dashrightarrow Y$ denotes the fact that the entity holds a recommendation for entity Y(allegedly) issued and signed by entity X. They use the word "‘alleged"’ because "‘without verification, there exists no evidence that the certificate was indeed issued by the claimed entity. Associated with each edge is a confidence parameter in the range from 0 to 1, which is assigned by the entity who creates it. Given a request to authenticate a public key, this work provides a metric to compute the confidence value in the range [0,1] for the key, using the confidence parameters associated with each edge in the certificate graph.

Similar to the model in [12], this model is also based on probability theory and the computed confidence values will depend on the confidence parameters associated with each edge. These confidence parameters are set by individual users and their accuracy will affect the accuracy of the computed confidence value.

2.5 Attack-resistant trust metrics for public key certification

In Levien and Aiken's work[42], it takes as input the set of digitally signed certificates which can be represented by a graph. In this model, there are *keys* and *names* and two types of certificates. A binding certificate asserts that a public key k belongs to a name n. A delegation certificate asserts the trust on certificates signed by a subject public key. A vertex in the certificate graph can either represent a public key or a binding of name and public key. A binding certificate is represented by an edge from the vertex corresponding to the issuer's public key to the vertex corresponding to a binding of name and public key. A delegation certificate is represented by an edge from the vertex corresponding to the issuer's public key to the vertex corresponding to the subject key.

In this work, two types of attacks are considered. In a node attack, the attacker is able to generate any certificates from the attacked key. In the certificate graph, this corresponds to that the attacker is able to add arbitrary edges from the vertex that represents the attacked key. This happens when the attacker obtains the private key material of the attacked public key. In an edge attack, the attacker is only able to generate delegation certificates from the attacked key. This happens when the attacker is able to convince the owner of the attacked key that an untrustworthy key can be trustworthy. This work analyzes different types of attacks on certificate graph and present the following network flow metric to authenticate public key.

Let s be the source vertex, i.e., the vertex corresponding to the public key of the user who evaluates this metric. Let t be the target vertex, i.e, the vertex corresponding to the binding of name and public key that the user needs to authenticate. In general, each node n will be assigned a capacity $C_{(s,t)}(n)$ based on specific rules and the $M(G, s, t)$ function is used to calculate the maximum network flow from s to t. The computed value of function $M(G, s, t)$ is then compared to a predefined threshold $\theta(s)$. Iff $M(G, s, t) \geq \theta(s)$, the binding of name and public key represented by s will be accepted.

The capacity of each node is computed as,

$$C_{(s,t)}(n) = max(f_s(dist(s, n)), g_t(dist(n, t))),$$

where $dist(s, t)$ is the length of the shortest path through the graph from s to t. Let $succ(s)$ be the set of successors of vertex s, the functions f_s and g_t that are designed specifically to resist node attacks are,

$$\begin{aligned} f_s(l) &= \begin{cases} max(\frac{1}{d}, \frac{1}{|succ(s)|}) & \text{if } l = 1 \\ \frac{1}{d} & \text{if } l \geq 2 \end{cases} \\ g_t(l) &= \frac{1}{d} \end{aligned}$$

d is the incoming degree for each vertex and is assumed to be constant in this work. Equations to compute f_s and g_t for edge attacks are also given, please refer to [42] for details. With the capacity $C_{(s,t)}(n)$ assigned to each vertex n in the certificate graph, standard maximum flow algorithms[5] can be used to compute $M(G, s, t)$ and be compared to $\theta(s)$ to determine the acceptance of the binding of name and public key.

This works also shows that it is efficient compared with other trust metrics, such as [45] and [55]. It does an excellent job on showing different types of attacks. Two different models are also presented to counter against these attacks. One comment we have on this work how to set the right value of threshold $\theta(s)$. Setting a higher value of threshold will reject more bindings of name and public key. This work recommends setting a threshold that will accept most bindings, for example 99%, assuming that most bindings are correct. However, it is unclear how effective this setting is. We have no idea on the rate of false positives and false negatives nor do we know what percentage of correct bindings will be rejected and what percentage of correct bindings will be accepted. Users have to be smart enough to set the correct threshold based on the level of attacks that the system is subject to.

2.6 Resilient authentication using path independence

In [53], the authors use more general terms, such as principles, channels, and statements to describe the problem of a principal to authenticate a channel. In the context of public key authentication in webs of trust systems, principles correspond to entities; channels correspond to public keys; and statements correspond to certificates. The problem of a principal to authenticate a channel is the same as the problem of an entity to authenticate a public key. In the following, we will use the terms of entities and public keys in webs of trust system to describe the approaches used in this work. The input is a directed certificate graph with vertexes as public keys and edges as certificates. Let s be the public key of the user who will evaluate the metric. Let t be the target public key to be authenticated. To authenticate a public key, the method will try to find a maximum set of Bounded Disjoint Paths(BDP) from s to t in the certificate graph. "'Bounded"' means that each path's length should not exceed a predefined value b. "'Disjoint"' means that no two paths share the same public key. The authors point out that the problem to find BDP is shown to be NP-hard[34]. The authors discuss the possibilities and difficulties of the approximations and present their own approximation algorithm. In their approximation algorithm, they first convert the BDP problem to the MIS(Maximum Independent Set) problem. The MIS problem is also a known NP-hard problem[29]. The authors suggest to use the simple approximation algorithm[37] to solve the MIS problem and the BDP problem. Using this approximation algorithm, the authors run some empirical experiments on PGP keyrings to justify their algorithm's performance.

The BDP model of authenticating public keys has a clear meaning in terms of countering against malicious attacks. If the number of BDP from s to t is 9, and the number of attacked public keys in these disjoint paths is smaller than 9, it is straightforward to see that we are able to authenticate the public key t. The reason is no matter where these attacked keys are, there is always a path from s to t where no attacked keys exist. Thus we can authenticate the public key t with 100% assurance.

We'd like also to note that length bound used in the BDP model. Without length bound, the problem to compute the maximum set of disjoint paths from s to t can be solved by standard network flow algorithms[5]. The intuition to use a length bound seems to come from the following observation. That is, the probability that a public key will be attacked in a shorter path will be lower than that in a longer path. [42] has a different opinion about this. Their opinion is that whether a name-to-key binding is correct or not has nothing related to the path length. We think that path length might be useful in some cases but it is not a deterministic factor to authenticate a public key.

In addition to BDP, the authors also use BCP(Bounded Connective Paths) as another means to authenticate a public key. The problem is to find the b-connectivity from s to t and the corresponding set of connective paths. Instead of directly proving that the problem of BCP is NP-Complete, the authors prove that determining if BCP $< k$ is NP-Complete. An approximation algorithm is also given for computing BCP.

Unlike other methods, this method doesn't require users to specify any trust on others. And the result it produces is very meaningful. If there are b public-key disjoint paths from public key k to public key k', then the owner of k may determine that k' is an authentic public key if the number of malicious users on these paths is less than b. The method will take as input a direct certificate graph, the source public key(the public key of the user who performs the trust computation), the target public key(the public key that needs to be evaluated for its authenticity), and output a value to represent the target public key's authenticity. We believe that this method is very convenient for users and represents a better way to serve users needs of authenticating public keys.

2.7 Authentication metric analysis and design

In addition to the model presented in [53], the authors have a second model for public-key authentication which is presented in [54]. They first present several principles of authentication metric design and then use these principles to analyze some authentication metrics, including [65], [12], and [45].

They also present a new model to authenticate public keys. In this new model, the concept of insurance is introduced in the public-key authentication. Basically, the input is still a certificate graph with vertexes as public keys and edges as certificates. Associated with each edge is a value in dollars, called insurance. The insurance associated with an edge $K_1 \rightarrow K_2$ represents the amount of money that the owner of K_1 insures the attributes bound to K_2 and the behavior of K2. Specifically, the insurance value is the amount of money the owner of K_1 will be liable if the attributes bound to K_2 in the certificate are incorrect, or if K_2 is used illegitimately, whether intentionally or not. Given the certificate graph, a trusted source key K_s and a target key K_t as input, the model will return an amount for which the name-to-key binding is insured by computing the capacity of a minimum capacity cut from K_s to K_t. The minimum capacity cut from K_s to K_t can be computed using standard network flow algorithms in [5].

In the webs of trust systems, this is the first model to introduce insurance in the trust computation. By introducing the insurance, it makes users to be more liable and responsible for the certificates they are issuing and thus can help prevent malicious attacks. In practice, whether users are willing to accept this type of insurance is a question. Involving money cost in webs of trust system is new issue because one of the reasons that users adopt webs of trust is because that it is free. The effects of introducing insurance in webs of trust system remain a question and can only be told in practice.

2.8 Self-Organized Public-Key Management for Mobile Ad Hoc Networks

[15] considers the problem of public-key authentication in webs of trust systems in the mobile ad-hoc networks(MANET). In wired networks, there is usually an on-line central certificate repository where users can retrieve other's public keys. In this central certificate repository, there are a lot of public-key certificates. Accompanying the central certificate repository, there is also a key server that answers user's requests for public-key certificates. When a user submits a query of a public key, the key server will search the certificate repository and return the corresponding public-key certificate. By using this on-line central certificate repository, users will be able to retrieve any public-key certificates they need.

In the mobile ad-hoc networks, it is difficult for any single network node to serve as a central certificate repository due to the limited power and bandwidth of each network node. In addition, the frequent movement of MANET nodes makes the communication situation harder and can create network partitions. Moreover, a node may leave the network intentionally or unintentionally (computer malfunctions, batter power or reset). All these situations make it difficult to have a single node serve as a central certificate repository in the MANET. To overcome these issues, the authors suggest the following approach. Each node in the network will choose a subset of all certificates and store them in its local certificate repository. When two users needs to communicate with each other, they will merge their certificate repository and try to find a certificate path between them.

The method to choose and get the set of certificates in each node's local certificate repository is described as follows: It has two phases. One is outgoing certificate collection and the other is incoming certificate collection. In the outgoing certificate collection, the node starts from itself and follows the path in its outgoing certificate chains. It runs in multiple rounds. In the first round, the node selects one of the certificates that it issued to others. Suppose the certificate is issued to user X_1, in the second round, it will query X_1 about the certificates the X_1 issued to others and choose one of them. The process continued until the number of certificates selected reaches the bound β. The selection of certificate is determined by the degree of the node that the certificate is issued to, i.e., each time we need to select a certificate, we will select the one that is issued to the node with the maximum degree (the number of certificates issued to others). These selected certificates represent a certificate chain of length β starting with the node itself. The method can also select certificates in multiple certificate chains, in the same procedure. The process of incoming certificate collection works in similar way, but in the reverse direction of certificate chains. These selected certificates construct the node's local certificate repository. Each node in the network will perform the same procedure to construct their local certificate repositories. When two nodes want to communicate securely with each other, they merge their certificate repository and try to find a certificate chain between them. Experiments were done to show the effectiveness of this method.

The presented method is suitable for MANET because of the following reasons. First, it is a distributed approach and doesn't rely on any central nodes to work. For two nodes to authenticate each other's public keys, the needed certificates can all be found local certificate repositories of each node. Second, no global information is needed. In the process of collecting certificates from others, each node will only need to query certificates that are issued to/from several other nodes. There is no need to query all the certificates issued by all nodes in the network. It is very efficient and saves communication bandwidth. Third, the maximum degree algorithm is a smart algorithm in choosing the right certificate. Its intuition is very straight-forward where the node with maximum degree has a better chance to connect to other nodes via certificate chains. Thus the chance to find certificate chains between two nodes is also increased by selecting such nodes.

Chapter 3

Computing webs of trust

3.1 Introduction

In general, authentication is considered one of the objectives in the information security world. Until the mid-1970s, secrecy was generally recognized to be a necessary part of authentication. With the invention of several cryptographic techniques, such as hash functions and digital signatures, it has been shown that secrecy and authentication are not necessarily connected[5]. Applying these cryptographic techniques, public key infrastructures have been established, with authentication one of the provided functions. Some examples are X.509 [60] and PGP [65]. In the public key infrastructure, each user is associated with a public key, which is publicly available, and with a private key, which is kept secret by the user. A user signs a message with her private key, and this signature can be authenticated using the user's public key.

The ability to exchange public keys securely is a very important requirement for the authentication scheme to work. Various ways may be used to exchange the public keys. For example, certificates are considered to be a good way to deliver public keys, and are popularly used in today's public key infrastructures. Intuitively, a certificate is an authority telling about a user's public key. In a hierarchical system, such as X.509 [60], we usually assume that the certificates contain true information because the authority is secured and trusted. The problem appears in a non-hierarchical system, such as PGP [65], which is referred to as a "web of trust". In such a system, each user becomes an authority. It is risky to believe all certificates contain true information, because not all users are fully secured and trusted.

Accepting a false public key as true undermines the foundation of authentication. For example, a user Alice may incorrectly recognize a false public key k created by a *malicious* (i.e., untrustworthy, unreliable) user John as Bob's public key. Any confidential information for Bob encrypted with k would be readable by John, which is undesirable. Bob's ability to authenticate is also compromised because John could generate digital signatures which would make Alice believe the information comes from Bob. For this reason, a method that can be used to securely verify a user's public key is very much needed.

The main goal in our work is to develop a robust scheme to determine if a certificate is trustworthy. Our method uses redundant information to confirm the reliability of a certificate. Previous work [53] has shown that redundancy in the form of multiple, independent certificate chains can be used to enhance reliability. We build on this work, but show that in some circumstances multiple certificate chains do not provide sufficient redundancy. This is because a single malicious user may possess multiple public keys, or (falsely) claim multiple identities, and, therefore, can create multiple certificate chains which seem to be independent. Our solution to this problem is to also consider identities when making use of multiple certificate chains.

In addition, it has previously been observed that a user may possess multiple public keys and conflicting certificates may occur, but the consequences of this possibility have not been explained. Conflicts are easy to detect. We show how this conflicting information can be used to narrow the list of *suspects*, or possibly malicious users. Based on that information, the number of certificates which can be proved to be true is increased, improving the performance of our method.

The organization of this work is as follows: In section 3.2, the notation used in this work and the assumptions of our method are described precisely. The research problem is defined, and the criteria to evaluate possible solutions are also discussed. Section 3.3 presents our solutions without considering conflicting certificates and the corresponding algorithms. Section 3.4 shows how conflicting certificates information may be used to improve the performance of our solutions in section 3.3. In section 3.5, we present the results of experiments on both synthetic data and PGP keyrings, and discuss the performance of our solutions. The last section summarizes our results and suggests future work.

3.2 Problem Description

A *user* is an entity in our system represented by an *identity*, such as the names "Bob" and "Alice". An identity must be established when the initial trust information is negotiated between users. We assume in this work that each user legitimately has exactly one, unique *true identity.* An identity which does not belong to a real user is a *false identity.* We consider the possibility in the following that a user may forge multiple, false identities, but there are not multiple true identities for each user.

We further assume each user can have, or be associated with, one or more public keys. In the case where a user has more than one public key, we assume the user further specifies each of her keys by a *key index number.* The combination of a user identity x and a key index number j is denoted x/j, and uniquely identifies a public key. If the user with identity x only has a single public key, j will be omitted, for the sake of convenience.

A *public key certificate* and a *certificate chain* are defined as follows, based on [48]:

Definition 1 *A public key certificate is a triple $\langle x/j, k, s_{k'} \rangle$, where x is an identity, j is a key index number, k is a public key, and $s_{k'}$ is the digital signature over the combination of x/j and k.*

Note $s_{k'}$ is associated with the certificate but k', the public key that could be used to verify $s_{k'}$, is not necessary in the certificate.

Given a certificate $C = \langle x/j, k, s_{k'} \rangle$, if (i) the identity x is a true identity, and (ii) the user with identity x says k is her public key[41], then C is a *true certificate* and k is a *true public key* for x. Otherwise, C is a *false certificate* and k is a *false public key* for x. If $s_{k'}$ is generated by y, we say the certificate is *issued* by y. If all certificates issued by y are true certificates, then y is a *good user.* If there exists at least one false certificate issued by y, then y is a *malicious user.*[1]

Two certificates are said to *agree* with each other if the identities, key index numbers, and public keys are the same. Two certificates are called *conflicting certificates* if the identities and key index numbers are the same but the public keys are different. Note that the two conflicting certificates may both be true by our definition (the user with the corresponding identity says both of the two keys are her public keys). This may happen when a

[1]We ignore the fact that a false certificate may be mistakenly issued with no malicious intent, for purposes of this paper. The method of detecting mistakes is the same as the method of detecting deliberately false information, so no distinction is required.

user x intentionally has two conflicting certificates issued to herself, by two separate parties, for the purpose of possessing more public keys. We expand our definition of a malicious user to also include x in such a case; the other parties, however, are not considered to be malicious on this count, and the certificates are defined to be true.

For a user w to decide a certificate C which contains identity x is false, C must conflict with another, true, certificate, and w must believe that x wouldn't ask for conflicting certificates to be issued to herself.

Each user x may accumulate a set R^x of certificates about other users. Obtaining these certificates may be done in a variety of ways, and may be volunteered (i.e., the *push* model), or may be provided on request (the *pull* model). For example, certificates can be obtained by retrieving from a (trusted) key server or by transmission directly from one user to another in a network. We do not discuss further in this paper how certificates are distributed; which is an open problem.

In each user's set of certificates, some of them are assumed by x to be true and others are not. Denote T_0^x the set of certificates assumed by x to be true initially (i.e., they are provided by means of the initial trust distribution). Because this initial trust information is assumed to be true, the signatures on the certificates in T_0^x do not have to be further verified. We now define a *certificate chain*:

Definition 2 *A certificate chain is a sequence of certificates where:*

1. *the starting certificate, which is called the* tail *certificate, is assumed or determined to be true;*
2. *each certificate contains a public key that can be used to verify the digital signature associated with the next certificate in the sequence; and,*
3. *the ending certificate, which is called the* head *certificate, contains a desired name-to-key binding, which is called the* target*.*

Each user x's set of certificates R^x may be represented by a directed *certificate graph*[2] $G^x(V^x, E^x)$. V^x and E^x denote the set of vertexes and the set of edges in the certificate graph G^x, respectively. A vertex in V^x represents a public key and an edge in E^x represents a certificate. There is a directed edge labeled with y/j from vertex k' to vertex k in G^x if and only if there is a certificate $\langle y/j, k, s_{k'}\rangle$ in R^x.

[2]Certificate graphs are also referred as *trust graphs* in some other research papers.

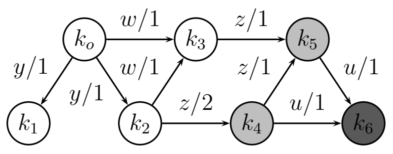


Figure 3.1: User x's certificate graph

A certificate chain is represented by a directed path in the certificate graph. Two conflicting certificates are represented by two edges with the same label, but different head vertexes (i.e., different keys for the same identity/index number). In this case, the two different head vertexes are called *conflicting vertexes.*

For computation purposes, we add to the certificate graph an "oracle" vertex k_0. There is a directed edge from k_0 to every key which is assumed to be true (i.e., by way of the initial trust distribution), labeled with the identity/index number bound to that key. A certificate graph only contains those public keys that are reachable from k_0; the public keys that are not reachable from k_0 are omitted.

To depict true and false public keys in the certificate graph, we will paint vertexes with different colors. A vertex painted white represents a public key that is either assumed or determined to be true for the identity on the edges incoming to that vertex. A vertex painted a medium gray represents a public key which is known to be false for the corresponding identity. If a public key is not assumed to be true and hasn't been determined to be true for the corresponding identity, we paint it with a light gray color.

Figure 3.1 is a sample certificate graph. k_0 is the oracle vertex. k_1, k_2 and k_3 are three public keys that are assumed to be true by user x. k_1 and k_2 are two conflicting vertexes because the labels on their incoming edges are the same. k_5 is z's public key, with index number 1, and k_4 is z's public key, with index number 2. k_6 is a false public key.

When there is more than one malicious user, there may be a relationship between these users. Two malicious users who cooperate with each other to falsify information are said to be *colluding.* In the case where it is unknown if two users have such a a relationship, a conservative assumption is that they are colluding. We say that two users x and y are colluding if either of the following is true:

1. There exist two false certificates, one issued by x and one by y, and they agree with

each other[3] ; or,

2. x issues a false certificate upon y's request, or vice versa.

Now we give a formal problem description and our criteria to evaluate the performance of any solutions. The problem statements are introduced gradually. First we consider the case in which there is only one malicious user. When there are multiple malicious users, we consider two cases, one in which the users are assumed not to collude, and one in which they do. The goals for these three problems are the same, that is, to determine the maximum number of true certificates.

PS 1 *Given a set R^x of certificates and a set $T_0^x \in R^x$ of true certificates. Assuming there is* at most one malicious user*, maximize the number of certificates which can be proved to be true.*

PS 2 *Given a set R^x of certificates and a set $T_0^x \in R^x$ of true certificates. Assuming there are* at most n malicious users*, and these users do* not collude*, maximize the number of certificates which can be proved to be true.*

PS 3 *Given a set R^x of certificates and a set $T_0^x \in R^x$ of true certificates. Assuming there are* at most n malicious users *and these users* may collude*, maximize the number of certificates which can be proved to be true.*

It is necessary for us to develop a criteria to evaluate the performance of proposed solutions for the above problems. In that way comparisons can be made among different approaches. Let U be the set of all users. Denote by K_a^x the set of true public keys that can be reached by at least one path from the oracle vertex k_0 in the certificate graph. K_a^x is the maximum set of true public keys that any method could determine to be true by means of certificate chains. Let K_0^x be the set of public keys that are assumed to be true initially, and K_M^x the set of public keys that are determined to be true by a method M. Ideally, K_a^x would be equal to $K_0^x \bigcup K_M^x$, but in practice this may be difficult to achieve. The fraction $\frac{|K_0^x|+|K_M^x|}{|K_a^x|}$ expresses the performance of a given method:

[3]We ignore here the very small possibility that two non-colluding malicious users could by chance create identical false certificates. Since keys are generated randomly and are quite long, the probability of independently generating the same key to put in two false certificates is extremely low.

Definition 3 *Given a solution s to a problem in 1, 2 and 3 generated by a method M. The performance $q_x(s)$, i.e. s's performance for user x, is*

$$q_x(s) \;=\; \frac{|K_0^x| + |K_M^x|}{|K_a^x|}$$

To capture the performance for a set of users, rather than just one, we propose using the weighted average of each user's performance:

Definition 4 *Given each user x's performance $q_x(s)$ for solution s generated by method M, the performance $q(s)$ of solution s for the set of users in U is the weighted average of the performance of s for each user. The weight of a user x is $|K_a^x|$, i.e., the number of public keys in x's certificate graph which are true.*

$$q(s) \;=\; \frac{\displaystyle\sum_{x \in U} \left\{ q_x(s) \cdot |K_a^x| \right\}}{\displaystyle\sum_{x \in U} |K_a^x|}$$

Definition 3 expresses the fraction of true public keys that are assumed or can be determined to be true for each user. Definition 4 expresses the fraction of true public keys that are assumed or can be determined to be true for all the users. Note that the above criteria match with our problem definition, i.e. the maximum set of true certificates corresponds to the maximum set of true public keys.

We now present methods for solving problems 1, 2, and 3.

3.3 Maximizing the number of true certificates when there are no conflicts

In this section, we present methods for solving problems 1, 2, and 3 under two assumptions (single or multiple keys per identity). We use redundancy, as have others, to confirm certificates that must be true. As mentioned, we assume there is a known upper bound on the number of users who may be malicious. We ignore in this section the case in which there are conflicting certificates, which is considered in section 3.4.1. We begin with some basic definitions and insights.

Two certificate chains are *public key independent* if

1. their head certificates agree; and,

2. their remaining certificates have no public keys in common.

Two certificate chains are *identity-independent* if

1. their head certificates agree; and,
2. their remaining certificates have no identities in common.

Theorem 1 *Given two* identity-independent *certificate chains, if there is at most one malicious user, the head certificates of the two chains must be true.*

Proof: We prove Theorem 1 by contradiction. The head certificates of the two chains, by definition, agree with each other. Suppose the head certificates were false. Then in each of the certificate chains, the malicious user's true identity must appear in the chain (i.e., the malicious user has participated in creation of the chain in order to insert a false certificate). In this case, however, the same identity would appear in each chain, violating our assumption that they are identity-independent. □

For multiple non-colluding malicious users, an analogous result holds:

Theorem 2 *Given two identity-independent certificate chains and any number of* non-colluding *malicious users, the head certificates must be true.*

Proof: The head certificates of the two chains, by definition, agree with each other. If the head certificates were false, each chain (for the same reason as above) must contain the true identity of a malicious user. Since the chains are identity-independent, and we assume one user has only one true identity, these malicious users must be different. But if the head certificates are false and they agree, these two malicious users must have been colluding, which violates the assumption. □

In the case that there are multiple colluding malicious users, a greater degree of redundancy is needed to verify a certificate is true:

Theorem 3 *Given $n + 1$ identity-independent certificate chains, if there are at most n colluding malicious users, then the head certificates must be true.*

Proof: We prove Theorem 3 by contradiction. Suppose the head certificates were false. Then in each of the $n + 1$ certificate chains, there must be a malicious user's true identity. Since there are $n+1$ identity independent certificate chains, so there must be $n+1$ malicious users, which violates our assumption. □

Based on these results, we now present methods for maximizing the number of true certificates (when there are no conflicting certificates).

3.3.1 One public key allowed per identity

Reiter and Stubblebine [53] are the first to consider this problem. We summarize their results.

When each identity corresponds to only one public key, public key-independent certificate chains are also identity-independent. In the certificate graph, public key-independent certificate chains corresponds to paths with the same tail vertex and the same head vertex, but which are otherwise vertex disjoint. For each vertex k_t, it is possible to use standard algorithms for solving maximum network flow [5] in a unit capacity network to find the number of vertex-disjoint paths to k_t from k_0.

It can be shown that:

1. if there is at most 1 malicious user, and the maximum flow from k_0 to k_t is greater than or equal to 2; or,
2. if there are any number of non-colluding malicious nodes, and the maximum flow from k_0 to k_t is greater than or equal to 2; or,
3. if the number of (possibly colluding) malicious users is no greater than n and the maximum flow from k_0 to k_t is greater than or equal to $n + 1$,

then all certificates (edges) ending at k_t must be true. This procedure is run once for each vertex k_t in G^x to maximize the number of certificates known to be true. The algorithm for solving maximum flow in unit capacity networks runs in $O(|V||E|)$.

In this paper, we do not consider the case in which certificate chains are restricted to be no longer than a specified bound, which is solved heuristically by [53].

3.3.2 Multiple public keys allowed per identity

We now address the case in which an identity may be associated with multiple public keys in x's certificate graph. The method of [53] does not apply here. For example, in figure 3.1 there are two vertex-disjoint paths from k_0 to k_6. If it is assumed there is at most one malicious user, the method of [53] will conclude that k_6 is u's true public key. However, suppose that user z has two public keys, k_4 and k_5, and this is allowed (legitimate).

In this case, it is not safe to conclude that k_6 is u's true public key, since z could be lying. This issue is also pointed out in [53], but is not addressed there.

We still wish to use the notion of redundant, identity-independent paths to overcome the influence of (single or multiple, colluding or non-colluding) malicious users. To ensure that two paths are *identity-independent* under the new assumption, it is necessary that the two paths have no label in common on their edges. In this case the paths are said to be *label-disjoint.* In this context, when we refer to the label on an edge, we use only the identity of the certificate issuer, and ignore the key index number.

Suppose there exists a solution to the problem of determining the maximum label-disjoint network flow in the graph G^x with unit capacity edges, from k_0 to a vertex k_t. We conclude (by the reasoning previously given) that k_t is a true public key for the identity on the edges in the maximum flow ending at k_t if and only if:

1. the maximum flow is 2 or greater, and there is at most 1 malicious node, or any number of non-colluding malicious users; or,
2. the maximum flow is $n + 1$ or greater, and there are at most n colluding malicious users.

We now state a theorem about the complexity of finding the maximum label-disjoint network flow in a directed graph:

Theorem 4 *Given a certificate graph G^x and a vertex k_t. If one identity may legitimately be bound to multiple public keys, the problem of finding the maximum number of label disjoint paths in G^x from k_0 to k_t is NP-Complete.*

We sketch a proof that a known NP-Complete problem, 3-CNF-SAT[17], is transformable to problem 4. The 3-CNF-SAT problem is described as follows. A boolean formula is in 3-CNF, or 3 conjunctive normal form, if it is expressed as an AND of clauses, each of which is the OR of three literals. A literal in a boolean formula is an occurrence of a variable or its negation. For example, $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee x_4 \vee x_5)$ is in 3-CNF form. It contains three clauses, each of which contains three literals.

Given a boolean formula ϕ in 3-CNF form, we are asked whether it is satisfactory. Let a *conflicting* pair of literals be a literal in one clause which appears in complementary form in another clause (for example, a literal x_1 appearing in one clause and $\neg x_1$ appearing in another clause). For each such conflicting pair, let there be a unique label assigned,

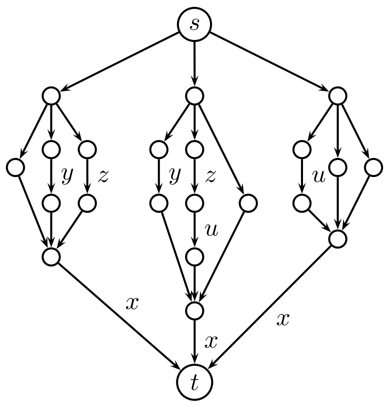


Figure 3.2: Graph constructed from the Boolean expression $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee x_4 \vee x_5)$

and let the number of conflicting pairs be represented by m. We construct a directed flow graph G from ϕ as follows. Each clause in 3-CNF-SAT is represented as a subgraph of G. In each subgraph, there are three paths. Each path corresponds to a literal in the clause, and has as many edges as there are conflicting literals in the other clauses of ϕ. Each such edge is labeled with the unique label assigned for this conflicting pair. The subgraph has a source vertex and a sink vertex, and the three paths in the subgraph are each connected (in parallel) to this source and sink.

After constructing in this way as many subgraphs as there are clauses, a vertex s is created to be the source for the graph G, and a vertex t is created to be the sink of G. s is connected by a directed edge to the source of each subgraph, and the sink of each subgraph is connected by a directed edge to t. Each edge that connects the sinks of all the subgraphs to the vertex t is assigned the same label, which must be different from all existing edge labels.

If any edges remaining in G do not have labels, each such edge should be assigned a unique label.

For example, in figure 3.2 we show the transformation from a 3CNF formula $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee x_4 \vee x_5)$ to the graph G by the method just described. There are three pairs of conflicting literals: x_2 in the first clause with $\neg x_2$ in the second clause, x_3 in the first clause with $\neg x_3$ in the second clause and $\neg x_3$ in the second clause with x_3 in the third clause. An edge is generated for each conflicting pair, and they are assigned the labels y, z and u respectively. For all other edges in G, since each edge has a unique label, the labels are not shown.

Given this graph G, it is possible to compute the maximum number of label disjoint paths from s to t by some algorithm. If the result is m, then ϕ is satisfactory, otherwise, ϕ is not satisfactory. Since the transformation to G requires only polynomial time, computing the maximum label-disjoint flow in the graph G with unit capacity edges is NP-complete. $\square$

Therefore, to solve this problem exactly will be very expensive computationally, in the worst case. We present two possible approaches.

We first consider an approach following an idea from [53]. The problem of finding the maximum number of label-disjoint paths between k_0 and k_t can be transformed to the maximum independent set (*MIS*)[29] problem. The *MIS* problem is defined as follows: Given a graph G, find the maximum subset of vertexes of G such that no two vertexes are joined by an edge. The transformation from our problem is trivial. From the certificate graph G^x, let each path from k_0 to k_t be transformed to a vertex in a new graph G^{MIS}. If two paths in G^x have a label in common, then the two corresponding vertexes in G^{MIS} are joined by an edge. Otherwise there are no edges in G^{MIS}. The size of the maximum independent set in G^{MIS} is the maximum number of label-disjoint paths from k_0 to k_t in G^x.

Although *MIS* is also a NP-complete problem, there exist several well-known approximation algorithms for it. See, for instance, [38] and [37].

Alternatively, we can consider solving the problem optimally if the required number of label-disjoint paths is small. Suppose we wished to solve the problem of whether there exist at least b label-disjoint paths in a graph from a source to a sink. The maximum number of label-disjoint paths from k_0 to k_t equals the size of the minimum label-cut for k_0, k_t. A label-cut is a set of labels on edges whose removal would disconnect k_t from k_0. The following enumeration algorithm 1 will determine the minimum label cut in a graph, up to a size of b. The algorithm runs in $O(|E||V|^b)$. The algorithm takes the certificate graph G^x and a target public key k_t as input, and outputs i, the size of the minimum label-cut for k_0, k_t.

In this section we considered how to prove certificates are true without reference to the possibility of conflicting certificates. We now turn to this problem.

algorithm 1 *label disjoint*

```
1: Delete all vertexes from $G^x$ that can't reach $k_t$.
2: set $L$=the set of labels on edges in $G^x$
3: for $i = 1...b$ do
4:   for each subset $L_i$ of $i$ labels in $L$ do
5:     set $G$=$G^x$
6:     delete each edge from $G$ which has a label in $L_i$
7:     if $k_t$ is not reachable from $k_0$ in $G$ then
8:       return $i$
9:     end if
10:  end for
11: end for
```

3.4 Maximizing the number of true certificates when there are conflicts

We assume that conflicting certificates occur because of malicious intent, and not by accident. A malicious user may create conflicting certificates for several reasons. For example, one use is to attempt to fool user x into believing a false public key is true, by creating multiple public key-independent certificate chains to the false public key. In this case, the method of section 3.3.2 can first be applied to determine the set of true certificates.

However, we can exploit the existence of conflicting certificates to prove an even larger number of certificates must be true. Conflicting certificates were mentioned in [53]. Although it was pointed out that conflicting certificates represent important information, it was not suggested how they could be used. We propose below a method of doing so, based on the notion of the *suspect set*:

Definition 5 *A suspect set is a set of identities that contains at least one malicious user. A member of the suspect set is called a suspect.*

We now describe how the suspect set can be constructed, and how it helps to determine more true certificates.

3.4.1 Constructing suspect sets (single or multiple non-colluding malicious users)

Suppose we know or have determined a certificate is false by some means. If there is only one malicious user, or multiple non-colluding malicious users, the true identity of the issuer of this false certificate must appear in a certificate of every chain ending with a certificate that agrees with this false certificate.

Using this insight, we propose constructing suspect sets using algorithm 2. The algorithm takes a certificate graph G^x as input.

Algorithm 2 *suspect set one*

```
1: for each label $y/j$ in the certificate graph do
2:   for each medium gray vertex $k_i$ whose incoming edge has a label $y/j$ do
3:     construct a new suspect set consisting of the set of labels, each of which is a label-cut for $k_0, k_i$.
4:   end for
5:   for each light gray vertex $k_i$ with an incoming edge labeled $y/j$, conflicting with a white vertex with an incoming edge labeled $y/j$ do
6:     construct a new suspect set consisting of {the set of labels each of which is a label-cut for $k_0, k_i$} $\cup\ y$
7:   end for
8:   for each pair $k_i, k_h$ of light gray vertexes whose incoming edges both have a label $y/j$ do
9:     construct a new suspect set consisting of {the set of labels each of which is a label-cut for either $k_0, k_i$ or $k_0, k_h$} $\cup y$
10:  end for
11: end for
```

The intuition behind algorithm 2 is as follows.

Lines 2-4: For each false certificate, the true identity of the malicious user who issued this false certificate must be in a certificate in *every* certificate chain whose head certificate agrees with this false certificate. This means, in the certificate graph the malicious user's true identity must be a label-cut for k_0, k_t (where k_t is the public key in the false certificate). Note there may be multiple such label-cuts.

Lines 5-7: Let y/j be a label which has been found in a true certificate. An undetermined certificate conflicting with this true certificate, may itself be true or false. If it is true, the malicious user must be y. If it is false, the identity of the malicious user who issued this false certificate must be in every certificate chain whose head agree with this certificate. In this case, the suspect set contains every identity which is a label-cut for k_0, k_t, plus y, where k_t is the public key in this undetermined certificate.
Lines 8-10: Let y/j found in two or more undetermined certificates. For each pair of conflicting certificates contain y/j, they may both be true, one may be true but not the other, or they may both be false. The suspect sets can be constructed for each case following the same intuition above, and the union of them is taken to get a single suspect set.
The complexity of this algorithm is O($\mid V \mid^3 \mid E \mid$). This algorithm may generate a large number of suspect sets. We now explain how this information can be used.

3.4.2 Exploiting suspect sets (single or multiple non-colluding malicious users)

Suppose there is a single malicious user. Let L_s represent the set that results by intersecting all the suspect sets generated by the above algorithm. Then clearly the single malicious user's identity is in L_s.

If, on the other hand, there may be up to b non-colluding malicious users, we try to determine the maximum disjoint sets (*MDS*) from all the suspect sets generated by algorithm 2. Two sets are called disjoint if they don't share any label in common.

Unfortunately, *MDS* is also NP-Complete, by transformation from the maximum independent set problem, MIS. First, each vertex in MIS is transformed into a set which contains just one distinguished label, representing that vertex. Then, for each edge joining two vertexes in MIS, a distinguished label is added to the two sets corresponding to those vertexes. A maximum disjoint set for this problem is also a solution for the transformed MIS problem. As before, we can apply known heuristics for MIS to solve this problem approximately.

Suppose a solution to MDS consists of m suspect sets, and $m = b$. Let L_m be the union of these m sets. It is clear that all b malicious users must be in L_m.

Given L_s or L_m, we can determine more certificates are true as follows. For each undetermined public key k_t, if there is only one malicious user, we simply test if any single label in L_s is a label-cut for k_0, k_t in the certificate graph. If not, k_t is determined to be true. If there are multiple non-colluding malicious users, we simply test if any single label in L_m is a label-cut for k_0, k_t. If not, k_t is determined to be true. For this computation, a modified BFS or DFS search suffices. The complexity for the algorithm is $O(|V||E|)$ for both cases.

3.4.3 Suspect sets (multiple colluding malicious users)

In the case of multiple colluding malicious users, we propose to use the following rules to construct suspect sets. These rules are presented in decreasing order of priority.

suspect set rule 1 *Given a certificate chain whose head certificate is false, construct a new suspect set that contains all the identities (except the identity in the head certificate) in the certificates of the chain.*

suspect set rule 2 *Given two certificate chains whose head certificates are conflicting with each other, if one of the head certificates is true and the other is undetermined, construct a new suspect set that contains all the identities in the certificates of the chain whose head certificate is undetermined.*

suspect set rule 3 *Given two certificate chains whose head certificates are conflicting with each other, if both head certificates are undetermined, construct a new suspect set that contains all the identities in the certificates of the two chains.*

The suspect sets constructed by these rules correspond to certificate chains. The number of suspect sets constructed can be very large because there could be a large number of different certificate chains. For example, even for a single pair of public keys, the number of certificate chains that connects them could be many. Also, the suspects in one suspect set can be overlapped with the suspects in other suspect sets. It may be very expensive and not necessary to construct all of them.

To make use of the many suspect sets constructed by these rules to determine more true certificates, we try to find the maximum number of disjoint sets (*MDS*) from all the suspect sets. Suppose the number of maximum disjoint sets is found to be a.

Let L_c be the union of the a sets. It is clear that at least a malicious users are included in L_c. Delete all the edges with a label in L_c from the certificate graph. By doing this, the maximum number of malicious users with certificates in the certificate graph is reduced from b to no more than $b - a$. In this case, Theorem 3 can be applied to determine if the rest of the undetermined certificates are true, as follows. For each target public key k_t, if there exist $b - a + 1$ label-disjoint paths between k_0 and k_t, the head certificates of these paths are true. Algorithm 1 can be used to solve this problem.

In this section we have explained how to generate suspect sets, containing the identities of malicious nodes. We described how to use this information to prove additional certificates must be true. We now present experimental results about the use of conflicts.

3.5 Experimental results

To investigate the practicality and benefits of our conflict detection method, we implemented and tested it. Only results for the case of one legitimate public key per user are available at this time. We emulated typical malicious user behavior, in order to contrast the performance before and after conflict detection.

For test purposes, we used actual PGP keyrings. These were downloaded from several PGP keyservers, and the keyanalyze [58] tool was used to extract many strongly-connected components. There are two reasons that we use strongly-connected components as the test data. One is that the PGP dump that we downloaded contains thousands of public keys, which is too large for our experiment algorithm to use. The other reason is that strongly-connected components usually represent a group of user's public keys. In addition, we synthetically generated keyrings to have a larger number of test cases to investigate. The synthetic data was generated by the graph generator BRITE [46], which we believe to be a very good PGP graph generator as it can generate Barabasi graphs that simulates social relationships[9]. The undirected graphs generated by BRITE were converted to directed graphs by replacing each undirected edge with two directed edges, one in each direction. The number of vertexes in each synthetic key ring was set to 100. For each data point, 50 problem instances were generated (using a different random number generator seed each time); the values plotted are the average of these 50 instances.

In our first experiment, we compare our result with the method of [53] on one of the largest PGP keyrings (whose graph contains 15956 vertexes and 100292 edges). On this PGP keyring we emulated the behavior of n colluding malicious user as follows. First we randomly picked a target, and n malicious users. The n malicious users issued n false

Number of colluding malicious users	1	2	3	4
Before conflict detection	78	54	40	33
After conflict detection	9992	77	53	40

Table 3.1: Results for PGP keyring before and after conflict detection for a total of 10,000 public keys

certificates, one per malicious user, each binding the target's identity to the same false public key.

After emulating this behavior, we applied the method of [53] to determine the maximum set of true certificates. The results are shown as *before* conflict detection. Then we applied the suspect rules of colluding malicious users (from section 3.4.3) to find many suspect sets, from which we constructed L_c. For each of the remaining undetermined public keys, we made use of L_c and the method of section 3.4.3 to try to determine if it was true. The results are shown as *after* conflict detection.

Table 3.1 shows the results. For this very large PGP keyring, it is not practical to compute the result for all users. Instead, we randomly picked 200 users. For each user, we randomly selected 50 public keys on which to test our method. Each user's public key graph was the entire keyring. The figure shows how many total public keys can be determined to be true when there are different numbers of malicious users. From this figure, it may be seen that the result for 1 malicious user is greatly improved (by two orders of magnitude) when conflict detection is used. In these cases, the suspect set turned out to be quite small; excluding the members of this set allowed many more certificates to be proved true. It may also be seen that the result with conflict detection drops dramatically (by the same amount) if there are two or more colluding malicious users. The reason for this is that this particular trust graph is not richly connected. There tends to be only one certificate chain between each pair of vertexes, which makes it difficult or impossible to find two or more independent paths to the target key after the suspect set is removed.

In our second experiment, we used the synthetic data for the case in which each user issues on average 2 certificates. We emulated a single malicious user's behavior, as follows. We randomly picked a target, a malicious user, and n certificate issuers. Then the

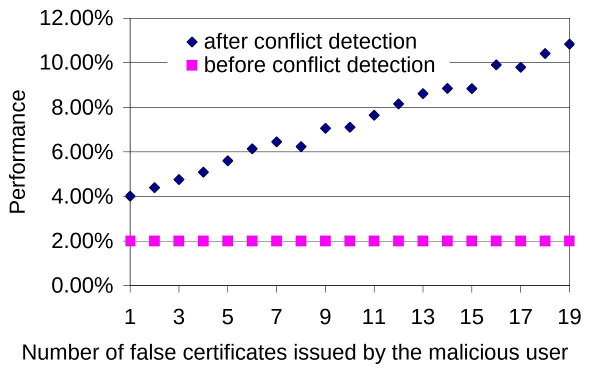


Figure 3.3: Improved performance for a sample synthetic data set with average 2 certificates after conflict detection

malicious user asked the n certificate issuers to certify n different public keys for them self. Using these n different public keys, the malicious user created n certificates, one per key, each binding the target's identity to the same false public key.

After emulating this behavior, we applied algorithm 1 for $b = 2$, and determined the maximum set of true certificates. The resulting performance is the performance *before* conflict detection. Then we applied algorithm 2 to find many suspect sets, from which we constructed L_s. For each of the remaining undetermined public keys, we made use of L_s and the method of section 3.4.1 to try to determine if it was true. The resulting performance is the performance *after* conflict detection. Each test was run 50 times and the averages are plotted.

Figure 3.3 shows how the performance q is affected by the number of false certificates n issued by a single malicious user, from a performance of 2% with 1 false certificate, to a performance of 11% with 19 false certificates. This example illustrates a dilemma for the malicious user. While increasing the number of false certificates should increase the uncertainty about keys, it also makes it easier to narrow the list of "suspicious" users, thereby limiting the scope of the damage they can cause.

Figure 3.4 is for existing PGP keyrings, and demonstrate how performance is improved by conflict detection. In each PGP key ring, we emulated a single malicious user's behavior as follows. We randomly picked a target, a malicious user, and two certificate issuers. Then the malicious user asked for two different public keys, certified by the two certificate issuers. Using these two public keys, the malicious user created two public key-

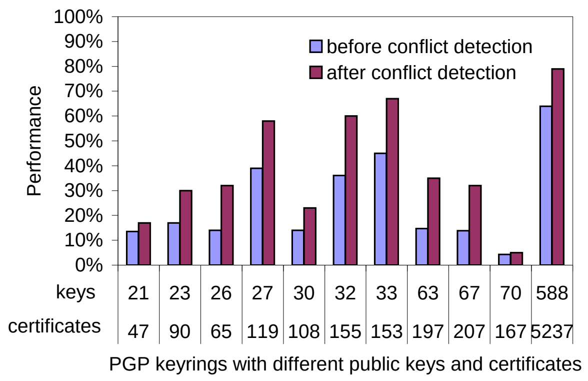


Figure 3.4: Performance comparison for PGP keyrings before and after conflict detection

independent certificate chains to the target. After emulating this behavior, we applied algorithm 1 with $b = 2$ to determine the maximum set of true certificates. The resulting performance is the performance *before* conflict detection. Then we generated L_s by algorithm 2. By making use of L_s, we tried to determine more true certificates. The resulting performance is the performance *after* conflict detection. The bar chart shows a comparison of performance for the case before conflict detection and after conflict detection for some PGP keyrings.

All of the experiments were performed on a Pentium IV, 2.0GHZ PC with 512MB memory. For the experiments on synthetic graphs with 100 vertexes, running times varied from 5 to 3 seconds. For figure 3.1, running times takes 1 to 2 hours. For figure 3.4, running times ranged from 5 to 30 seconds, except for the graph with 588 vertexes, which required 10 hours of CPU time. Our implementations are in no way tuned for efficiency and can undoubtedly be substantially improved. We believe in most environments the algorithm for detecting true certificates will not need to be run frequently, and a running time of hours will be acceptable.

3.6 Conclusion

In this work, we investigated the problem of proving certificates are true when trust is distributed, as in PGP keyrings. This is a difficult problem because malicious users may falsify information. Under the assumption that users may legitimately have multiple public keys, we showed that redundant *identity-independent* certificate chains are needed. Previous methods based on *public key-independent chains* are not sufficient under this assumption. In the case that certificate conflicts are detected, it is possible to exclude certain users from the set of possible malicious users. This allows additional certificates to be proved true. Experimental results demonstrated that (a) the web of trust is seriously degraded as the number of malicious users increases, and (b) the use of conflict detection and redundant certificates substantially improves the ability to prove certificates are true.

We have already shown the method to measure the robustness of webs of trust system. In the next chapter, we will show how to improve this robustness.

Chapter 4

Improving webs of trust

4.1 Introduction

In the previous chapter, we have discussed a lot of methods that can measure the robustness of web of trust. However, they do not address how to enhance it. In this chapter, we will show that the practical web of trust provides very poor robustness on user-provided key information. This poor robustness is the motivation for this work. We adopt the model of [53] for measuring the assurance of user-provided key information. We believe this model is more suitable for the case of malicious users. It is much easier to bound the number of users who may be malicious, than to specify precisely how trustworthy each user is.

Our method of enhancing the robustness of web of trust requires some users to issue additional, recommended certificates. We hope the user would comply with this request in most cases, but recognize that users may have willingness and preferences about the issuance of certificates. For instance, some users may wish to limit the number of certificates they issue, or may only agree to issue certificates to selected individuals. An incentive for users to issue the recommended certificates is that the robustness of whole web of trust will be enhanced by doing so. The robustness represents a level of confidence for the entire community against malicious users' attacks, thus representing a benefit to all.

There are mainly three contributions in this work. First we present a close-to-optimal heuristic to solve the problem of enhancing robustness in distributed web of trust. The heuristic is to *enhance* the robustness of web of trust entirely in the most efficient way. The output of our algorithm is a set of recommendations to users for whom additional certificates should be issued to. Second, for these results to be practical, we consider how users' willingness and preferences on following our recommendations may affect the robustness of the resulted keyrings, respectively. By users' willingness we mean a user may follow only part of our recommendations. For users' preference, we use the popular notion of buddy lists and emulate it using the small world model which is frequently observed in PGP and social networks. The third contribution is the application of our recommendation method can enable users to detect malicious attacks and identify users who may be malicious.

This work is organized as follows. Section 4.1 introduces some background information and related works on using redundancy to measure the robustness of web of trust. Section 4.2 defines the research problems and our assumptions. The criterions for evaluating the proposed solutions are also discussed. Section 4.3 describes an algorithm for efficiently enhance the robustness of web of trust. Section 4.4 illustrates the performance of our solutions on both actual and generated web of trust using experiments. Section 4.5 investigates the impact of users' willingness and users' preference on the recommendations, respectively. Section 4.6 discusses and shows how our recommendation method may be used to help users to detect attacks and identify malicious users. The final section concludes our work.

4.2 Problem Statement

The notations used in this chapter are basically the same as those in chapter 3. We don't repeat them in this chapter.

In all methods of authenticating public keys [59, 47, 12, 45, 52, 53, 42], multiple certificate chains represent high assurance for the name-to-key binding in question. More specifically and accurately, [53] has shown multiple disjoint certificate chains is an essential measurement of the assurance of name-to-key bindings. The problem of enhancing the robustness of web of trust is thus very straightforward. That is, how to create required number of disjoint certificate chains for all users and all name-to-key bindings. If there are q vertex-disjoint paths in the certificate graph, each starting from user x's own key and all ending at the same target name-to-key binding, the *assurance* of the target name-to-key binding for x is defined to be q. That is, x would be able to say that the target name-to-key binding is resistant to attacks on less than q keys as shown in [53, 42]. The number of vertex disjoint paths can be computed by a maximum flow algorithm using an vertex-splitting technique[5]. There are various maximum flow algorithms available, such as those in [5].

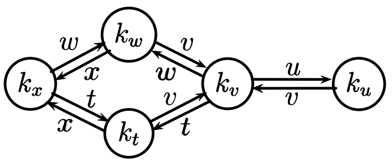


Figure 4.1: A sample certificate graph

Figure 4.1 is a sample certificate graph. In this graph, there are ten arcs corresponding to ten certificates. Users can determine the assurance of name-to-key bindings using the vertex-disjoint paths in the following way. For user x, bindings w/k_w and t/k_t are correct because they are certified by x directly. Binding v/k_v has an assurance of two because there are two vertex-disjoint paths from k_x to k_v. However, for user x, assurance of u/k_u is one, because there is only one vertex-disjoint path from k_x to k_u. Following the same idea, for user u, binding v/k_v is correct, and the assurance of all other name-to-key bindings is only one.

To measure the assurance of all name-to-key bindings for each user x, we introduce the following notions in a certificate graph. Denote $K_x(G)$ the set of all *correct* name-to-key bindings whose keys are reachable from key k_x(x's own key) in the certificate graph G. Denote $K_x^T(G, l)$ the set of name-to-key bindings whose assurance are greater or equal to a predefined threshold l (for user x, binding x/k_x and the name-to-key bindings x directly certifies are correct, thus their assurance is set to be infinite). For user x the *robustness* of the certificate graph G at assurance l is defined to be

$$\mathcal{A}(x, G, l) = \frac{\mid K_x^T(G, l) \mid}{\mid K_x(G) \mid}$$

We define the total, or aggregate, robustness of a certificate graph G at assurance l for all users to be

$$\mathcal{A}(G, l) = \frac{\sum_{x \in U} \mid K_x^T(G, l) \mid}{\sum_{x \in U} \mid K_x(G) \mid}$$

It is straightforward to see this robustness represents the percentage of name-to-key bindings whose assurance is at least l, thus robustness can range from a minimum of 0% (worst) to 100% (best).

We can now formally state the problem addressed by this paper. The goal is the following. Given a directed certificate graph G and a value l representing a desired level of assurance, add the minimum number of arcs (i.e., additional certificates) to this graph so that robustness $\mathcal{A}(G, l)$ of whole certificate graph reaches 100%.

Certificates are created by users, and in most systems those users are not under central control. For this reason, we refer to the certificates added by a solution to the above problem as certificate *recommendations*. We return to the issue of users' willingness and preferences and their impact on recommendations at the end of this paper. The next section presents a method for accomplishing the goal.

4.3 Enhancing Certificate Graphs

From the preceding discussion, assurance is enhanced when the number of vertex-disjoint paths between pairs of vertexes is increased. By Menger's theorem, vertex-disjoint paths correspond to graph connectivity, specifically, a graph is q-connected iff every pair of vertexes is joined by at least q vertex-disjoint paths, or to say, there does not exist a set of q or fewer vertexes whose removal disconnects the graph [30]. The key requirement in achieving our goal is therefore to make the directed certificate graph G q-connected, where $q \geq l$ and l is the desired level of assurance for all users and all name-to-key bindings.

The problem of increasing the connectivity of a graph by adding a minimal number of edges is referred to as the *graph connectivity augmentation* problem. There are a number of solutions to this problem and a quite complete survey has been done both in [50] and [24]. Among the numerous solutions to this problem, only [25, 10, 26] are able to augment the vertex-connectivity in directed graphs. Because the certificate graph is directed and our goal here is to increase vertex connectivity, only these solutions would apply to the problem presented in this paper. Unfortunately all these solutions [25, 10, 26] are very expensive and complicated. For example, [25] relies on the ellipsoid method [49] which is well known to be impractical. [10] points out its practical efficient implementations require oracle choices. [26] requires a very expensive running time of $O(|V|^6 f(k))$ where $|V|$ is the number of vertexes, $f(k)$ is a super-exponential function of k and k is the required connectivity. Thus they are not suitable in practice.

To efficiently solve this problem, we propose a heuristic for the graph augmentation problem. Given is a certificate graph G and a required connectivity $q (q \geq l)$. The algorithm outputs a graph G' which is *enhanced* or augmented to have a connectivity q, which corresponds to 100% robustness at assurance level l. To describe the heuristic, we first introduce some notations. Let $G = (V, E)$ be the given certificate graph. The *in-degree* $\Delta_i(v)$ of a vertex v in V is defined as the number of arcs in G whose terminal is v, while the *out-degree* $\Delta_o(v)$ of v is defined as the number of arcs in G whose initial is v. Let $\lambda_i^q(v)$ be the in-degree *q-deficiency* of vertex v, i.e., the difference to be added to make $\Delta_i(v)$ greater than or equal to q, and $\lambda_o^q(v)$ is similarly defined for the out-degree. The *minimum degree* of a graph G is represented as $\delta(G)$, and is defined as

$$\delta(G) = min\{\Delta_i(v), \Delta_o(v) | v \in V\}$$

The heuristic has two phases, and is shown as Algorithm 4. In Phase 1, arcs are added between vertexes with the smallest in- or out-degrees. The purpose is to increase $\delta(G)$ to reach q by adding a minimum number of arcs. In Phase 2, each vertex's in- and out-degree is at least q but G may not be q-connected. To make G q-connected, algorithm 3 is used to detect critical pairs of vertexes whose connectivitys are lower than q, then arcs are directly added between them.

Algorithm 3 q-connectivity algorithm

```
input: digraph $G$
output: vertex connectivity $q(G)$
1: select a vertex $u$ with the minimum $\{\Delta_i(u) + \Delta_o(u)\}$
2: compute $c_1 = \min\{q(u,v) \mid v \in V - \{u\}, (u,v) \notin E\}$
3: compute $c_2 = \min\{q(v,u) \mid v \in V - \{u\}, (v,u) \notin E\}$
4: compute $c_3 = \min\{q(x,y) \mid x \in P(u), y \in O(u), (x,y) \notin E\}$
5: $q(G)$=min{$c_1$, $c_2$, $c_3$}
```

Because algorithm 3 is called inside algorithm 4, we will explain algorithm 3 before we give details on algorithm 4. We first introduce some notations. Denote the vertex connectivity between two nodes u and v as $q(u, v)$. The vertex connectivity $q(u, v)$ between u and v can be computed by a maximum flow algorithm [5] using vertex-splitting techniques in time O($|V||E| \log(\frac{|V|^2}{|E|})$). The vertex connectivity $q(G)$ of G, is the minimum vertex connectivity of any pair of vertexes in graph G, i.e.,

$$q(G) = min\{q(u, v)| \text{ ordered pair } u, v, (u, v) \notin E\}.$$

$q(G)$ can be easily determined by computing the maximum flow for every ordered pair of vertexes in G and taking the minimum. However, we propose algorithm 3 as a faster

Algorithm 4 *graph connectivity augmentation algorithm*

```
input: $G(V,E)$, $l$
output: $G'(V,E')$
1: $G'(V,E') = G(V,E), q = l$
Phase 1
2: construct an arc list $L = \{(u,v) \mid u \in V, v \in V, (u,v) \notin E', (\Delta_o(u) < q \vee \Delta_i(v) < q)\}$
   ordered by $\mathrm{SUM}(\Delta_o(u) + \Delta_i(v))$
3: while $\delta(G') < q$ do
4:    choose the first arc $e$ in $L$
5:    add arc $e$ to $E'$
6:    update $L$
7: end while
Phase 2
8: run algorithm 3 and construct the arc list $L = \{(u,v) \mid q(u,v) < q,\ u \in V,\ v \in V\}$
   ordered by $q(u,v)$.
9: while $q(G') < q$ do
10:    choose the first arc $e$ in $L$
11:    add $e$ to $E'$
12:    run algorithm 3 and update $L$
13: end while
```

means of computing $q(G)$. This algorithm follows some insights from an approach in [5] for computing edge connectivity of a directed graph. The set $P(v)$ of predecessors of vertex v is defined as $P(v) = \{u|(u, v) \in E\}$, and the set $O(v)$ of successors of vertex v is defined as $O(v) = \{u|(v, u) \in E\}$.

We now explain algorithm 3. A *minimum vertex-separator* $S(S \subset V)$ is a set of vertexes with the least cardinality whose deletion from G would make $G - S$ disconnected. It is simple to see that $q(G) = |S|$. Suppose by some method we have found a pair of vertexes u and v such that $q(u, v) = q(G)$. We consider the following abstraction of a directed graph G and an arbitrary minimum vertex-separator S in G. In Figure 4.2, S is a minimum vertex-separator whose deletion would destroy all paths from u to v. L contains u and all vertexes reachable from u after deletion of S. R contains the remaining vertexes. Clearly, for all $w \in L$ and all $z \in R$, $q(w, z) = q(u, v) = q(G)$.

For an arbitrary vertex x, consider the following three possibilities for the location of x.

1. $x \in L$ (Line 2 in Algorithm 3). If there is at least one vertex y in R, then $q(G) = q(x, y)$. We simply compute the connectivity from x to all other vertexes. At least one vertex must be in R. This step requires to run maximum flow $|V|$ times.

$$q(G) = min\{q(x, y)|(y \in V - x) \wedge (y \text{ is not adjacent to } x \text{ in } G)\}.$$

2. $x \in R$ (Line 3 in Algorithm 3). If there is at least one vertex y in L, then $q(G) = q(y, x)$. We simply compute the connectivity from all other vertexes to x. At least one vertex must be in L. This step requires to run maximum flow $|V|$ times.

$$q(G) = min\{q(y, x)|(y \in V - x) \wedge (y \text{ is not adjacent to } x \text{ in } G)\}.$$

3. $x \in S$ (Line 4 in Algorithm 3). In this case, at least one of x's predecessors must be in L, and at least one of x's successors must be in R. We simply compute the vertex connectivity from all of x's predecessors to all of x's successors and choose the minimum one. Because x is chosen with the minimum degree and $\delta(G) = q$, $\Delta_i(x) \approx q$ and $\Delta_o(x) \approx q$. This step requires to run maximum flow approximately q^2 times.

To summarize, algorithm 3 is to check the connectivity of a set of $2|V| + q^2$ pairs of vertexes, and see if any pair's connectivity is lower than q. If none, G is already q-connected.

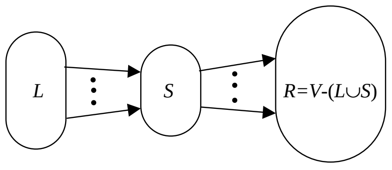


Figure 4.2: Graph $G = (V, E)$ and a minimum vertex-separator S

Otherwise, G would need more arcs added to become q-connected. The connectivity of each pair of vertexes is checked using maximum flow algorithm and algorithm 3 requires to run maximum flow $2|V| + q^2$ times.

By this point, the purpose of the application of algorithm 3 in algorithm 4 is very clear. If algorithm 3 tells us G is already q-connected, then algorithm 4 can simply stop because the goal of making G q-connected is already completed. Otherwise, algorithm 3 would tell us which pair of vertex has a connectivity lower than q, and our heuristic's choice is to simply add an arc directly between them. Now we explain algorithm 4.

In algorithm 4, phase 1 is to make $\delta(G) \geq q$ by adding a minimum number of arcs. It is simple to see that in a q-connected graph, any vertex's in- and out-degree must be at least q (We refer to this as the $q - degree$ requirement). Thus the number of arcs needed to be added to make $\delta(G) \geq q$ is at least

$$max(\sum_{v \in V} |\lambda_i^q(v)|, \sum_{v \in V} |\lambda_o^q(v)|).$$

Our heuristic successfully achieves this goal by adding an arc from the vertex with the minimum out-degree to the vertex with the minimum in-degree each time. In algorithm 4, line 2 constructs an ascending list L of vertex pairs ordered by the sum of the first vertex's out-degree and the second vertex's in-degree. Clearly, the leading pair of vertexes in L has the least in-degree and out-degree, respectively. To make sure no duplicate arc will be added, any pair of vertexes in L is checked to see if there's already an arc between them. Line 3 starts a loop by checking if the minimum degree of all vertexes is q. If yes, it will exit the while loop and go to phase 2. If no, line 4 and 5 choose the leading vertex pair in L and add an arc between them. Line 6 updates L because the degrees of the leading pair of vertexes in L were changed. Line 7 completes the loop. In phase 2, line 8 first runs algorithm 3 and constructs a list L of vertex pairs whose connectivity is lower than q. Line 9 starts a loop by checking if the graph's vertex connectivity is already q(if L is empty, then the graph is already q-connected; otherwise it is not). If yes, algorithm 4 stops because the goal of making the graph q-connected is already achieved. If no, line 10 and 11 choose the vertex pair with the least connectivity and add an arc directly between them. Line 12 run algorithm 3 again in order to update L and check if the graph is q-connected or not. Line 13 completes the loop.

Having explained the details of the heuristic, we now can discuss the quality of the presented heuristic. The heuristic is guaranteed to terminate because the number of arcs in a simple graph is finite and our heuristic adds no duplicate arc to it. The soundness of the heuristic, i.e. the graph produced will be q-connected, is provided by algorithm 3. The heuristic will not stop if algorithm 3 tell this goal is not achieved. The completeness, i.e. our heuristic will always find a solution, is straightforward because as the number of arcs increased in a simple graph, the graph will finally become a complete graph. In the last, we discuss the worst-case complexity of our heuristic. Phase 1 only needs to check and sort each vertex's degree. It requires $O(|V|log(|V|))$ time. In phase 2, suppose a total number of ε additional arcs are needed to make the graph q-connected, then algorithm 3 needs to be called ε times because it is called once for each additional arc. Algorithm 3 itself needs to run maximum flow $2|V| + q^2$ times. The running time of phase 2 is $O(\varepsilon \cdot |V|^2|E|\log(\frac{|V|^2}{|E|}))$ and the same for algorithm 4.

With this method, we can now address the goal of increasing the robustness of a certificate graph to 100%. For any arbitrary level of assurance l, the certificate graph is enhanced or augmented to be q-connected($q \geq l$) using the above method. The method is guaranteed to terminate, and the enhanced certificate graph is guaranteed to have an robustness of 100%. The arcs added by the method represent recommendations of certificates to add to the certificate graph. The delivery of these recommendations to users may occur the following two ways. The simplest way is to send these recommendations directly to the users by email. [1] It may also serve as a web service where each user would be able to submit a query and retrieve there lists of recommendations.

We have described heuristic methods for accomplishing our goal. In the next section, we examine the performance of the heuristic on actual web of trust: PGP keyrings. In the process, we will also make some observations about the original assurance provided by these PGP keyrings.

[1]In PGP, each user's email address can be directly obtained from their certificates.

KR25	PGP keyrings with 67 keys and 209 certificates
KR105	PGP keyrings with 105 keys and 245 certificates
KR588	PGP keyrings with 588 keys and 5237 certificates
KR1226	PGP keyrings with 1226 keys and 8779 certificates
SKR588	Synthetic keyrings simulating KR588
SKR1226	Synthetic keyrings simulating KR1226

Table 4.1: PGP keyrings used in this experiment

4.4 Experimental Results

The effectiveness and practicality of the proposed methods depends on the data to which it is applied. The best examples of webs of trust that are widely available are PGP keyrings[18]. There are many PGP keyservers, for example, pgp.cc.gatech.edu and wwwkeys.ch.pgp.net. The typical size of PGP keyring at each server is around 27,000 keys[58]. [2] The PGP keyrings we use in experiments are downloaded from these keyservers and the keyanalyze [58] tool was used to extract many strongly-connected components. In these strongly-connected components, the size of each keyring range from 20 to 16449 keys. The PGP keyrings used in this experiment are listed in table 4.1. Our method can be used to enhance any PGP keyrings to reach 100% robustness at any arbitrary level of assurance l. To demonstrate the effectiveness of our method, results are shown for chosen range of l, from 2 to 10, which we believe to be fairly good examples of levels of assurance. While KR588 and KR1226 are not the largest keyrings, they exercise the capabilities of our methods thoroughly, and it is possible to run large numbers of experiments on variations of these keyrings. Experiments were also run for those smaller keyrings, with similar results; these experiments are omitted for space reasons.

We first analyzed the robustness of the four actual PGP keyrings in table 4.1 for varying levels of assurance l. The results are shown in Figure 4.3. Two of these keyrings exhibit very low robustness at very low assurance $l > 1$. A third keyring (KR588) has better robustness, but also drops to a low level for $l > 5$. Only the smallest keyring, KR25, shows a higher level of robustness at high assurance values. This analysis provides strong evidence that large actual webs of trust do not achieve high levels of robustness. We believe this demonstrates the need for recommendations to increase the robustness of web of trust against attacks.

[2]A good place to retrieve PGP keyserver information is the mailing list PGP-keyserver-folk.

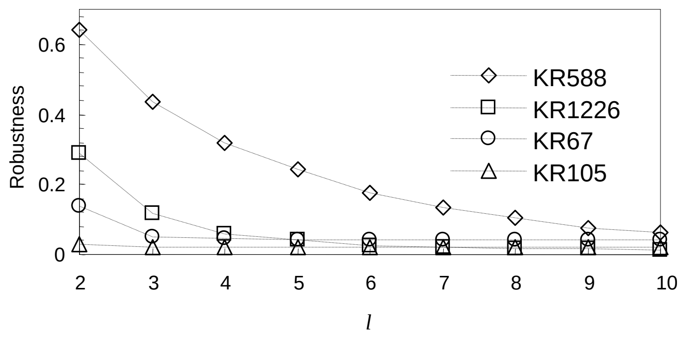


Figure 4.3: Robustness of some actual PGP keyrings

In the second set of experiments, we attempted to enhance the robustness of the actual keyrings KR588 and KR1226. As mentioned, these are some of the larger connected components in the PGP keyring repositories. We used algorithm 4 to enhance its robustness to 100% for varying levels of assurance l. The performance of the proposed method is shown in Table 4.2. We compared the performance with a method which randomly adds arcs to the graph. Any reasonable heuristic should, of course, perform better than the random method. We also computed a lower bound for the number of arcs which must be added by *any* method in order to achieve q-connectivity. It is very intuitive to see that in a q-connected graph G, each vertex's in- and out-degree must be greater or equal to q. By this requirement, it is straightforward to see that the minimum number of arcs that must be added to make G q-connected is at least

$$max(\sum_{v\in V}|\lambda_i^q(v)|, \sum_{v\in V}|\lambda_o^q(v)|).$$

The performance of our method (number of arcs added) relative to the random method, and to this lower bound, is shown in Table 4.2 for the actual keyrings KR588 and KR1226. It is clear that our heuristic performs much better than the random method. In addition, our method is very close to optimal for these two keyrings; in all cases, the difference between the lower bound and our method is no greater than 0.7%.

To increase confidence that these results are representative, we also synthetically generated keyrings using the model of [16]. This keyring generator will take the degree distribution of a given certificate graph and then generate many synthetic certificate graphs which have similar degree distributions. In this experiment, we generate two sets of keyrings,

PGP keyrings	Different methods	Total number of arcs needed at different levels of assurance								
		$l = 2$	$l = 3$	$l = 4$	$l = 5$	$l = 6$	$l = 7$	$l = 8$	$l = 9$	$l = 10$
KR588	lower bound	5343	5523	5754	6029	6351	6706	7085	7484	7907
	our method	5349	5524	5754	6030	6352	6710	7086	7484	7907
	random method	8168	8877	9864	10505	11959	13991	14654	14901	15267
KR1226	lower bound	9147	9720	10403	11187	12034	12931	13846	14781	15744
	our method	9203	9769	10458	11223	12061	12960	13876	14827	15796
	random method	17369	20385	24679	24709	26717	31507	32350	33352	33372

Table 4.2: Number of certificates needed to reach 100% robustness for PGP keyrings

Synthetic keyrings	Different methods	Total number of arcs needed at different levels of assurance								
		$l = 2$	$l = 3$	$l = 4$	$l = 5$	$l = 6$	$l = 7$	$l = 8$	$l = 9$	$l = 10$
SKR588	lower bound	5409	5598	5814	6077	6376	6713	7077	7469	7880
	our method	5409	5598	5815	6077	6377	6714	7078	7469	7881
	random method	9289	10255	11324	12376	13157	14219	15132	15841	16787
SKR1226	lower bound	9286	9874	10528	11284	12102	12973	13874	14808	15763
	our method	9287	9875	10530	11285	12103	12974	13875	14809	15763
	random method	18466	20989	23524	25923	27565	29635	31379	33448	35217

Table 4.3: Number of certificates needed to reach 100% robustness for synthetic keyrings

i.e. SKR588 and SKR1226, using the degree distribution of KR588 and KR1226, respectively. For each set of keyring, 50 synthetic keyrings were generated, based on the method of [16]; the variations in these 50 synthetic keyrings were due solely to the use of different random number seeds. The results are shown in Table 4.3. Each value shown in this graph is the average of 50 synthetic instances and has been rounded to the nearest integer, and has a 99% confidence interval of ±5 certificates. The results show our method consistently performs close to optimal, and much better than a random method. Our heuristic is not designed specifically for certificate graphs. It can also be applied generally to other kind of graphs and should produce similar performance.

These experiments also yield useful insight into existing PGP keyrings. First, it may be seen that "'most"'(additional certificates) to achieve 100% robustness for a lower level of assurance is small (e.g., less than 10% additional certificates with no more than assurance 4). However, the overhead grows steadily, and for a higher level of assurance (e.g., greater than 10) it may be expected this overhead is substantial.

Our experiments also show the practical running time for algorithm 4. In algorithm

4, phase 1 requires very little time to complete because what it does is just to check each vertex's degree and choose the minimum one. Phase 2 occupies most of the running time because algorithm 3 requires to call maximum flow $2|V| + q^2$ times. Practical experiment data tells us that algorithm 3 is called by algorithm 4 for only a few times. In all cases, ε is no greater than $\frac{|V|}{50}$. The heuristic is implemented in Java and C and the test environment is Linux Fedora Core 1 on a PC with a 2.4GHz Pentium IV processor and 512MB of memory. For the PGP keyring KR588, in all cases algorithm 4 requires less than 2 minutes to achieve a robustness of 100%. For the keyring KR1226, the corresponding times are 30 minutes. If these algorithms are executed off-line and infrequently, and on PGP keyrings of similar size, these running times are acceptable. If the method needs to be executed interactively, however, or for much larger PGP keyrings, increases in processing will be necessary.

4.5 User Preferences (Constraints)

In our discussion of the problem we termed the addition of certificates *recommendations*. This is an acknowledgment that certificates are generated by people, frequently for personal reasons. It may be expected that some users will be more willing than others to follow the recommendations generated by automated methods, such as those presented in this paper.

We propose that user preferences be incorporated into the method for enhancing certificate graphs, in the form of constraints. The many forms such preferences can take is outside the scope of this short discussion. We only intend to indicate two approaches and show that our heuristic is feasible even under such constraints. For these purposes, we propose to model user preferences as *user compliance* and *buddy lists*, respectively. By user compliance, we mean some users may choose to follow only part of our recommendations instead of all of them. This is our recognition that users may not wish, or be able to follow all of our recommendations. Buddy lists are another form to represent users' preference. They are frequently used in messenger systems, such as MSN, Yahoo Messenger and AOL Messenger. It happens that a user interacts more frequently with users on their buddy lists, thus it is a practical idea to use buddy lists as a constraint for users to issue certificates. To be specific, any user would not issue certificates to users other than those on their buddy list. Of course, a user may already have issued a certificate to another user in their buddy list before applying our method. In this case, our method would not recommend the user to issue

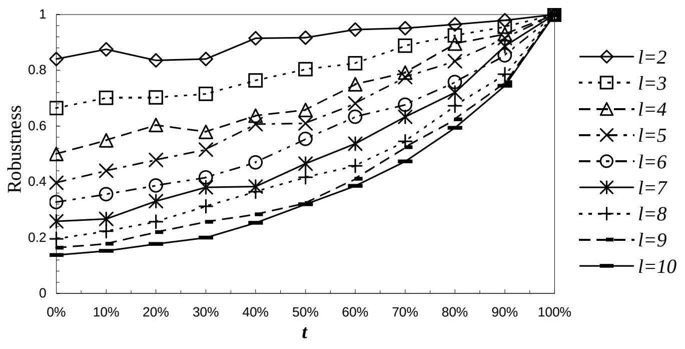


Figure 4.4: Robustness of KR588 under different percentage of users' compliance

a duplicate certificate. User compliance and buddy lists are different. In user compliance, the recommendation method remains unchanged but users are not supposed to follow all of the produced recommendations. It's the user's own choice to comply with part, all, or none of our recommendations. While in buddy lists, the recommendation method takes users' preference into account while making recommendations. All the recommendations are made under each user's buddy list constraint. One consequence of user compliance and buddy lists (and of user preferences in general) is that it is no longer possible to guarantee that the enhancement method will always terminate with a robustness of 100%. Experiments for these two types of user preferences have been done and are shown below. Only results for KR588 are available at this time.

User compliance. To investigate how user compliance may affect the performance of our method, we ran the following experiments. First we applied algorithm 4 and produced all the recommendations that are needed to reach a 100% robustness at assurance levels from 2 to 10. Having all these recommendations, t percentage of users are randomly chosen to be the set of users who will follow all of our recommendations. All the rest users are not supposed to follow all of our recommendations. Instead each user will randomly choose a part (from 0% to 99.99%) of our recommendations to follow. The results are shown in figure 4.4 for different values of t.

From this figure, we can see that for small levels of assurance($l < 5$), the robustness is always higher than the corresponding percentage of users' compliance. When assurance goes higher, the rate of increase drops and the robustness can't keep up with the percentage of users' compliance. One of the reasons is for lower levels of assurance, the number of recommendations is small, thus the users who don't comply with recommendations have little impairment on the resulted robustness of keyrings. It may be expected that the robustness is as least as t. But this goal may not be achieved due to the following reasons. By the $q - degree$ requirement, for a user x to have an assurance on key k, vertex k_x's out-degree and key k's in-degree must be at least q. Suppose a t percentage of users follow all of our recommendations, their out-degree would be at least t. To guarantee a robustness of at least t for the whole certificate graph, all other users' keys must also have an in-degree of q. Because some recommendations to fulfill this in-degree requirement are not followed, this in-degree requirement can't be met and the robustness of at least t may not be guaranteed. We also observed at some points of t, some of the curves in figure 4.4 drop when t increases. This is due to the randomness in selecting the t percentage of users who will follow all of our recommendations. At each point of t, the set of users who will follow all of our recommendations is randomly redrawn from the set of all users. This phenomena indicates that the set of users who will cooperate is also an important factor that affects the resulted robustness of keyrings.

Buddy lists. Practical data of buddy lists, such as MSN, Yahoo Messenger and AOL Messenger are not available to us for known reasons. Instead, we artificially generated the buddy lists by utilizing the small world graph generator of [16]. Small world phenomena have been frequently found in social networks [61], World Wide Webs [39] and PGP web of trust [16]. Specifically, the users' buddy lists may be represented by a world graph, where a vertex represents a user and an edge represents the buddy relationship. If there's an edge from u to v, then v is on u's buddy list. The small world graph generator of [16] takes as input a predefined degree distribution. We generate such a distribution using power-law as it has been frequently observed in small worlds [6, 16] [3]. Specifically, the size of each user's buddy list followed a power-law distribution, with exponent α = -1, a specified minimum value b, and a maximum value $\mid V \mid/10$.

The minimum value is enforced by simply discarding any randomly generated value less than b, and similarly for the maximum value. The selection of buddies is specified by the generator. We generated 9 different sets of user preferences (sets of buddy lists), for values of b from 2 to 10. For each set of user preferences, we measured the achievable robustness with our method at different levels of assurance from 2 to 10. Incorporating the constraint of buddy lists into algorithm 4 is simple. In this algorithm, when building the candidate

[3]It is pointed out in [16] that the degree distribution of PGP web of trust follows Zipf's law, which is shown to be equal to power law in [4]

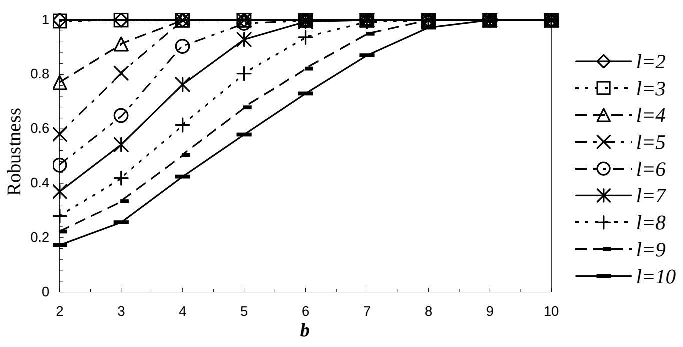


Figure 4.5: Robustness of KR588 under different buddy lists

arc list L, we simply exclude those arcs (u, v) where v is not in u's buddy list. The results for algorithm 4 with buddy lists constraints are shown in Figure 4.5 with a maximum input value of $l = 10$.

From this figure it can be seen that robustness increase roughly linearly with the increase of b, the minimum size of buddy list. The rate of increase of robustness decreases as b goes up. And a 100% robustness is achieved once b becomes the same as the level of assurance. There is little impact on the running time of the algorithm to incorporate user preferences such as buddy lists. The more significant impact is that users must be more flexible in their preferences (e.g. expand their buddy lists) to achieve high robustness in the requirement of increasing level of assurance.

4.6 Certificate conflicts and suspect set

Certificate conflict was first caught attention in [53] and being analyzed in [36]. Certificate conflict is defined to be a pair of certificates in which the two name-to-key bindings share the same name but different keys. The definition is extended to the case that each user may have multiple keys by simply associating a sequence number with each key[36]. Certificate conflict is very important because it is an indication of the existence of malicious behavior and may be used as evidence against malicious users. To analyze the impact of our recommendation method on certificate conflicts, we first briefly introduce the notion and methods on certificate conflicts that was presented in [36]. Certificate conflicts are created by malicious behaviors. For example, If an malicious user *Bob* try to cheat us to believe that *Alice*'s key is k but actually *Alice*'s key is k', then the two name-to-key bindings create a certificate conflict. Using the observation, certificate conflicts can be used to construct *suspect set* and detect malicious users. A suspect set is defined to be a set of names in which at least one belongs to a malicious user and may be constructed using the following rules.

1. If a name-to-key binding is found out to be incorrect, then a suspect set is constructed by including all names in any certificate chain from a known correct binding to the one before the incorrect binding.
2. If a name-to-key binding is unknown to be correct or not but conflict with a known correct binding, then it is either correct or incorrect and a suspect set is constructed by including all names in any certificate chain from a known correct binding to the unknown binding.
3. If two name-to-key bindings are both unknown to be correct or not but conflict with each other, then the possibilities of being both correct, one correct ,one incorrect, and both incorrect must all be considered. A suspect set is then constructed by including all names in any two certificate chains which both start from a known correct binding but end at the two unknown bindings, respectively.

The intuition of these rules is very straightforward. That is, at least one malicious user's name must appear in the certificate chain which ends at an incorrect name-to-key binding. A suspect set may be used to identify malicious users and clearly the smaller the suspect set, the better. If a suspect set contains only one member, then we have successfully found a malicious user.

An important impact of our recommendation method is that it helps in detecting certificate conflicts and identifying malicious users.

On the attackers' side, they would try to avoid certificate conflict by either certifying a incorrect name-to-key binding where the name has not yet appeared in any certificate or removing a certificate which contains the correct binding being attacked. But in our recommendation method, for each incorrect name-to-key binding, a set of specific users would be chosen to certify it. A certificate conflict could possibly be created because of the possibility that the *chosen* users are colluding with the specific malicious users are very small.

For example, suppose there is an incorrect name-to-key binding x/k certified by malicious user w, and our recommendation method suggests user y, z and v to recertify it. The probability that user y, z and v are both colluding with w should be very small because y, z and v are arbitrary chosen by our method. Assume the fraction of colluding malicious users in the network is 10%, then the probability that y, z and v are both colluding with w would be roughly 0.1%. By detecting certificate conflicts, users are enabled to know if the name-to-key binding of interest is being attacked. And users can focus their attentions and efforts on figuring out what the truthful information is. This information may be obtained from the corresponding user directly.

Another impact of the recommendation method is that the keyring would be made more robust and more name-to-key bindings would be identified to be correct, thus the size of each suspect set would be smaller and the chance to identify an attacker would be greater. If an attacker is successfully identified, as a punishment, we may be able to exclude the attacker from the community of web of trust, deleting all certificates issued and leaving them no chance of attacking in the future. The punishment may also serve as a mechanism of attack prevention. That is, the intention and interest of attack would be deterred because of the great possibility of being detected and punished.

To show actual impact of our recommendation method, we ran the following experiment by using KR588 as the test keyring. First, we created new name-to-key binding, randomly choosing a user to issue a certificate. This name-to-key binding represents the victim. Second, we created an incorrect name-to-key binding for the victim, i.e., a name-to-key binding with the same name as the victim but a different key. Third, we randomly chose 9 malicious users and had each of them issue a certificate for the incorrect name-to-key binding. Then we applied our recommendation method to this keyring with $l = 10$. Note in this experiment, all malicious users were not supposed to follow our recommendations simply because they might have not liked our recommendations. If users issued certificates as our method expected, the results were appealing. Before applying our method, no certificate conflicts could be detected and no suspect set could be constructed. After applying our method, all users were able to detect the conflicts and over 99% users were able to construct very small suspect sets. The size of each suspect set was only 3. The cost for each user in this experiment was approximately ten additional certificates. If users could go one step further, e.g. contacting the victim, and the name-to-key binding certified by malicious users turned out to be incorrect, then over 99% users would be able to construct

9 suspect sets and each suspect set contained only one member. That is, almost all users would be able to identify all the 9 malicious users. Of course, for a user to be able to do this, they must contact the victim and the victim must respond to it.

4.7 Conclusions

In this paper we described how distributed web of trust could be made resistant to attacks on keys by malicious users. Our approach required increasing the vertex connectivity of a certificate (directed) graph, which was a known problem with optimal (but complex) solutions. We presented a heuristic for this problem which ran quite efficiently for practical PGP keyrings. We applied these algorithms to current PGP keyrings. Our analysis showed that current PGP keyrings were highly vulnerable to attacks on keys by malicious users. Experimental results indicated that the robustness of PGP keyrings could be greatly increased with only a small increase in the number of certificates. In addition, we addressed the issue of user behavior and the constraints that adds. We investigated the impact of users' compliance of our recommendations on the robustness of PGP keyrings. We also demonstrated by experiments that PGP keyrings could be made very robust as long as user "buddy lists" were larger than the desired level of assurance. Finally, we raised the issue of certificate conflicts and identified malicious users. The application of our method enabled users to detect certificate conflicts and identify malicious users. Specifically, users were enabled to know whether their name-to-key bindings of interest were being attacked and if so, they could concentrate their efforts on resolving it. The application of our method could also serve as a determent to malicious attacks because of user's ability to detect these attacks and identify the malicious users.

We have described a certificate recommendation method to improve the robustness of webs of trust system. In the next chapter, we will present another method of certificate recommendation that is suitable for different assumptions.

Chapter 5

Constructing webs of trust

5.1 Introduction

In chapter 4, we have presented a certificate recommendation method that can improve the robustness of the webs of trust system. The method is convenient for users because it only requires users to set an upper-bound on the number of unreliable users. However, this method requires each user to issue an additional l certificates, if there are at most l unreliable users. If there are a lot of malicious users, then each user will be required to issue a large number of certificates. It makes it impractical for the method to work in this case.

The goal of this work is to find a much efficient method that can be used to improve the robustness of webs of trust in the face of a large number of malicious users. The differences of the recommendation method presented in this work and in chapter 4 are as follows:

In terms of soundness, they are both excellent but have slight difference. The recommendation method in chapter 4 provides 100% percent guarantee on the correctness of a name-to-key binding while random recommendation method in this chapter usually provides a probability guarantee on it. The probability guarantee demonstrated in the experiments is 99.99% but it can be tuned any desired number. In terms of efficiency, their differences are as follows. First, we examine the number of additional certificates needed by each of them. On average, the recommendation method in section 4 will approximately need l additional certificates for each name-to-key binding in a system with l unreliable users. For the random recommendation method in this chapter, each name-to-key binding will need approximately $\log_{\tau} 0.0001$ (given the probability is 99.99%) additional certificates to authenticate it in a system with τ percentage of unreliable users. Clearly, the random recommendation method in this chapter requires fewer additional certificates than the other recommendation method, especially when there are a large number of unreliable users. Second, the recommendation method in chapter 4 can produce recommendations based on buddy lists of a user, thus making the certificate recommendation process more user-friendly. For the random recommendation method in this chapter, the probability guarantee is based on the random generation of certifiers. Users don't play any role in choosing their certifiers.

The organization of the work is as follows. Section 5.2 will define the research problem precisely. Section5.3 will present our method of certificate recommendation to verify the correctness of a name-to-key binding. Section 5.4 will show how conflicting certificates information may be used to differentiate reliable users from unreliable ones and show how user's non-compliance may affect its effectiveness. Section 5.5 concludes the paper.

5.2 Problem Description

The notations and definitions are slightly different from those in the previous two chapters. We will introduce the differences as needed.

A set Π of certificates may be represented by a directed *certificate graph* $G(V, E)$. V and E denote the set of vertexes and the set of edges in the certificate graph G, respectively. A vertex in V represents a name-to-key binding and an edge in E represents a certificate. There is a directed edge from vertex (x, k') to vertex (y, k) in G if and only if there is a certificate $\langle y, k, s_{k'} \rangle$ in Π. A certificate chain is represented by a directed path in the certificate graph.

Figure 5.1 is a sample certificate graph. There are eight name-to-key bindings corresponding to eight certificates. For example, $(Mike, k_6)$ and $(John, k_5)$ are two name-to-key bindings and the edge from $(John, k_5)$ to $(Mike, k_6)$ represents the certificate $\langle Mike, k_6, s_{k_5} \rangle$

In the certificate graph, each name-to-key binding should be reachable from the binding of the owner, i.e., the user who will perform computation on the certificate graph. It is a general knowledge that an unreachable name-to-key binding should not be taken into consideration in the trust computation. Also in webs of trust systems, each user has their own point of view on the system and makes their own decisions based on the information

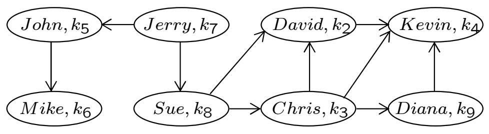


Figure 5.1: A sample certificate graph

they have. It is possible that different users will have different results of trust computation using the same method on the same certificate graph. It is simply because their views of the same certificate graph are actually different due the fact that their positions in the certificate graph are very different. With these notations and assumptions, we are now able to formally define the research problems as following:

PS 4 *Given a set* Π *of users, an upper-bound* τ *on the percentage of unreliable users and a probability* ζ*, design a method for users to certify any name-to-key binding such that its correctness can be guaranteed with probability* ζ*.*

Problem 4 defines the problem of public-key authentication in webs of trust systems in a different perspective from that in chapter 4. It considers the number of unreliable users as a percentage of all users. This makes it more scalable for large networks with large number of users. Also, the correctness of name-to-key bindings is based on probability but not 100% guarantee. Thus making it possible for us to develop a more flexible and efficient solution.

The definition of problem 4 reflects the requirements of both soundness and efficiency. Soundness is a must for any solution to this problem while efficiency is the factor that we need to optimize. Clearly, in this problem, we want to minimize the number of additional certificates needed for each certificate recommendation.

We have stated the assumptions and problem definitions. In the next section, we will present our solution to this problem.

5.3 Random Certificate Recommendation

In this section, we will present the method of random certificate recommendations to assure the correctness of any name-to-key binding. This certificate recommendation method follows some motivation of the method in chapter 4. They both try to increase of the robustness of webs of trust by issuing additional certificates. But they differ greatly in the selection of additional certificates to be issued. The method presented in this section has two very attractive features. First, it only requires moderate number of additional certificates to be issued even faced with a large percentage of unreliable users and/or in a very large network. Second, the result is very appealing since it can produce almost a 100% guarantee on the correctness of name-to-key bindings. These two features make the method an ideal candidate for systems with large number of unreliable users. The intuition behind our method is straight-forward. If multiple independent users certify the same name-to-key binding, then we should be highly assured about its correctness. Only in the case that all the certifiers of the same name-to-key binding are unreliable can it be incorrect and this case should rarely happen. For example, figure 5.1 is a certificate graph. Suppose Jerry, Sue and David certify the name-to-key binding of $(Mike, k_6)$. If not all of them are unreliable, then $(Mike, k_6)$ should be correct. The case that all of them are unreliable should be very rare if they are randomly chosen and recommended to certify this name-to-key binding. Under this method, we randomly choose and recommend users to issue certificates between them instead of leaving the unreliable users the choices to select a victim and collude to certify an incorrect name-to-key binding. We minimize the likelihood that users selected by our method are colluding with each other by producing the recommendation in a random fashion. That is, for each name-to-key binding in question, we randomly select some users and recommend them to certify the name-to-key binding. In the following, we will use probability theory to analyze our recommendation method.

Suppose that up to a percentage τ of all users in the system are unreliable. If we randomly pick up a set of β users from all λ users of Π and recommend them to certify a given name-to-key binding, the probability that these β users are all unreliable is simply $\frac{C^{\beta}_{\lambda \cdot \tau}}{C^{\beta}_{\lambda}}$. For example, if we pick up 4 users in a system where $\lambda = 100$ and τ=10%, the probability that these 4 users are all unreliable is less than 0.006%. In other words, we can guarantee that in over 99.994% cases, the recommendation should work and the certified name-to-key binding is correct. Denote ζ the threshold for the probability that the recommendation works. Simply, $1 - \frac{C^{\beta}_{\lambda \cdot \tau}}{C^{\beta}_{\lambda}} \geq \zeta$. The method is very easy to use and the probability guarantee is straight-forward. Provided with a desired probability ζ on the correctness of a interested name-to-key binding and an upper-bound on the percentage τ of unreliable users in the network, the method can produce a recommendation which includes a list of recommended users to issue certificates to make sure the correctness of the name-to-key binding.

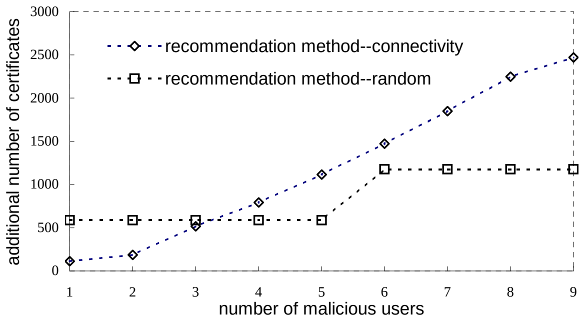


Figure 5.2: Comparison of two certificate recommendation methods

Our recommendation method is also very scalable. β grows in a log scale to the increase of λ, τ and the probability guarantee.[1] This is a very nice feature of our method because it makes it possible for the method to apply in very large networks and networks where there are a large percentage of unreliable users. For a network where $\lambda = 10^6$ and $\tau = 0.2$, if we set $\zeta = 99.99\%$, then we would only need $\beta = 6$ which is a very modest and practical requirement. Our recommendation method is very innovative in that it can guarantee the correctness of a name-to-key binding with a provable probability. This property makes it a very attractive solution for webs of trust systems that have high requirements of security. Other methods are able to perform computation on webs of trust but few of them can guarantee the correctness of their results with very high probability in the case of large networks and a lot of unreliable users.

We did a comparison between of this random recommendation method with the recommendation method presented in section 4.3. In this comparison, we use the same experiment data from the previous recommendation method and computed the required number of additional certificates for the random recommendation method. The PGP keyring used in this experiment is the one with 588 keys and 5237 certificates. We set $\zeta = 99.99\%$, $\lambda = 588$. The upperbound on the number of malicious users ranges from 1 to 9, τ is computed accordingly. With ζ, λ and τ, we can compute β. $\lambda \cdot \beta$ is the total number of additional certificates required by the random certificate recommendation method. We compare this result directly with the one presented in section 4.4.

In figure 5.2, X axis represents the upperbound on the number of malicious users in the network. Y axis represents the additional number of certificates needed by the

[1]Clearly the probability guarantee should be measured by the decrease of $1 - \zeta$.

certificate recommendation methods in order to verify the correctness of all name-to-key bindings in the network. The curves represent the two different recommendation method. The recommendation method–connectivity represents the previous recommendation method presented in section 4.3 and the recommendation method-random represents the random recommendation method presented in this section. In this figure, we can see that if the number of malicious users are less than 4, the previous recommendation method requires less number of additional certificates. However, this number grows in proportion of the number of malicious users. For the random recommendation method, it requires fewer additional certificates when the number of malicious users goes beyond 3 and increases very slowly with it. Clearly, the random recommendation method will require fewer additional certificates than the previous recommendation method. This shows that the random recommendation method is more useful when there is a large number of malicious users.

We have introduced our method of certificate recommendations to verify the correctness of a name-to-key binding. The best part of the method, as shown, is that it can enable users to be almost 100% sure about the correctness of a name-to-key binding in large scales at a reasonable cost. In the following, we will discuss some of the practical concerns about this method. First, as commonly recognized in all methods of computing webs of trust, issuing certificates requires the verification of identities. That is, we need to make sure the user is who they claim to be when issuing their the corresponding certificate. Verifying identities can be done in many ways. Usually it can be done by exchanging identity information, such as driver licenses, passports and etc. The identity verification may also be done by a trusted third party. Or they can call each other to verify the information. Another way is they can have their identity documents notarized and mailed to each other, which is actually one of the way being used by CAcert.org[1] for issuing free ssl certificates. Our recommendation method can use any of the identity verification methods including but not limited to those above. For example, when we generate a recommendation for a name-to-key binding and the user of this name-to-key binding receives is notified. The user can simply make β copies of their identity documents and have all of them notarized and mailed to the users in the recommend list. When the users in the recommend list receive these documents, they can simply verify the notarization and issue the corresponding certificates. These documents are usually assumed to be trusted as they are authenticated. We believe that it is a practical method because by doing so, the user contributes to the webs of trust system by issuing additional certificates, which represents a benefit to the entire community.

Also the requirement of showing proof of identity is not a exceptional requirement. In most public-key infrastructure systems, proof of identity is required when issuing a certificate. For example, in PGP, user may verify each other's identity by exchanging driver licenses.

The second issue of the method is user non-compliance. The method of certificate recommendations assumes that all recommended users will issue certificates as requested. But it may not be practical to expect all users to fully comply with all such requests. If some users don't comply with the requests produced by our recommendation method, then the probability guarantee can't be achieved. In consideration of this issue, we developed a solution which achieves the same goal on the condition that not all users will comply with our certificate recommendations. That is, some users may not choose to follow the recommendation that we have made and issue the corresponding certificates. It is obvious that malicious users may not choose to follow our recommendations simply because they never have that intention. Instead, they may choose to issue a false certificate to that user for the purpose of corrupting our method. While in this case, these false certificates will not make our method fail because in the worst case, they can be treated as non-compliance. And based on the probability analysis that we will shown below, as long as enough number of users certify the correct name-to-key binding, the recommendation method will still work. In the following, we would still like to address this issue using probability theory. Suppose we pick up a set ω of β users from λ users and only α users of them follow our recommendation, we are interested in finding out the probability that all the α users are unreliable. Clearly, if there are less than α unreliable users in ω, then it is not possible that any set of α users are all unreliable. We will only need to take care of the cases that there are α or more unreliable users in ω. First we calculate the probability that there are exactly x unreliable users in ω. Simply, for a set ω of β users, we will need to pick up x users from all unreliable users and pick up the rest $\beta - x$ users from all reliable users. The probability that there are exactly x unreliable users in ω is

$$\frac{C^{x}_{\lambda\cdot\tau} \cdot C^{\beta-x}_{\lambda(1-\tau)}}{C^{\beta}_{\lambda}}.$$

And the probability that there are α or more unreliable users in ω is

$$\sum_{x=\alpha}^{min(\beta,\lambda\cdot\tau)} \frac{C^{x}_{\lambda\cdot\tau} \cdot C^{\beta-x}_{\lambda(1-\tau)}}{C^{\beta}_{\lambda}}.$$

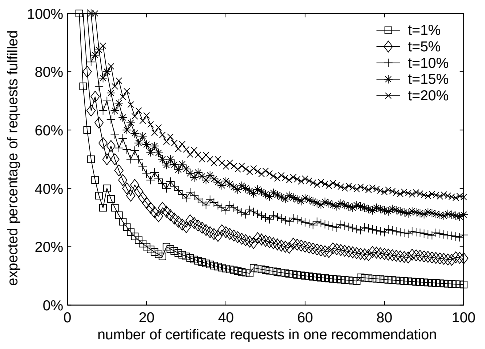


Figure 5.3: Percentage of user-compliance needed

And ζ should be set to

$$1 - \sum_{x=\alpha}^{min(\beta,\lambda\cdot\tau)} \frac{C_{\lambda\cdot\tau}^{x} \cdot C_{\lambda(1-\tau)}^{\beta-x}}{C_{\lambda}^{\beta}} \geq \zeta.$$

Given λ, ζ, β and τ, it is possible to compute α. Similarly, given λ, α, β and τ, it is possible to compute ζ. The fraction $\sigma = \frac{\alpha}{\beta}$ represents the minimal percentage of users who need to comply with the recommendation to make the probability guarantee work. Clearly σ is an important factor to measure the practicality of the proposed method. And usually say, the smaller the σ, the better chance for the method to work. To investigate the practicality of the method, we compute some example values of α for different β and show them in figure 5.3. In figure 5.3, λ is set to be 10^6, ζ is set to be 0.01%. We choose five different values of τ corresponding to the five difference curves. In figure 5.3, we can see that λ has little impact on σ. The major factor that affects σ is τ and β. With the increase of β, σ decreases. The rate of decrease is dramatic at first but slows down later. By increasing β, we actually reduce the requirement of user compliance, i.e., the method requires less percentage of users to comply with the recommendation to make it work. But increasing β to a very large number may not be a good choice because the actual number α increases too. It means that though the percentage of user compliance decreases, the actual number of users who need to comply actually increases. We have to be very careful in choosing β and the resulting σ. A small σ is usually more practical but a too small σ would make α very large and cost too much user's efforts. We suggest choosing those turning points shown in figure 5.3 when possible. These turning points are those in transition from dramatic decrease to slow decrease. For example, for the curve with $\tau = 1\%$, one of the turning point is at $\beta = 20$. Figure 5.3 also shows that our method is practical when there are a large amount of unreliable users. It only requires a modest number of users to comply even in a very large network. For a system with 1,000,000 users where 200,000 are unreliable, when $\beta = 12$, only 75% of users, i.e., 9 users are required to comply in each recommendation. And when $\beta = 40$, only 50% of users, i.e., 20 users are needed.

Note that it is possible that malicious users may create bogus ids in the webs of trust system. If the number of bogus ids are very large, then τ can be a large number which will increase α and β. One way to get around this problem is to make recommendations in a restricted set of users. The restricted set is constructed by eliminating those appears-to-be bogus ids. In this case, we will only pick up users from the restricted set and ask them to certify name-to-key bindings. λ should be set to be equal to the size of the restricted set. The user whose name-to-key binding needs to be certified is not subject to this restriction.

In the context of certificate graphs, where a vertex represents a public key and an edge represent a certificate, the random method that we have discussed in this section produces graphs that are different from "‘random graphs"’ in graph theory. In the graph theory, random graphs are constructed in the following way. Given a set of vertexes, random pick up pairs of vertexes and connect them. In our method, for each vertex, we will randomly pick up a fixed number of other vertexes and connect them. The obvious difference is that in our method, each vertex will have a fixed in-degree and similar out-degree because of the uniform random property. The random graphs in graph theory don't have this property because of the way it is constructed. Regarding connectivity, random graphs need approximately $n \cdot log(n)$ edges to reach one connectivity. Our graph should produce much better connectivity because of the similar degrees of vertexes.

We should also point out that for the random recommendation method to work, the recommendations need to be made by a trusted and authentic source. It can be a trusted central authority that makes these recommendations. And these recommendations should be verifiable when they are needed. A recommendation that is made by a untrusted party is not useful in ensuring the correctness of name-to-key bindings because of the possibility of malicious recommendations. If a malicious user makes a recommendation which includes all malicious users, then there's no reason to believe that the certificates issued by them contain the correct name-to-key bindings.

We have presented our simple yet very effective certificate recommendation method. We also discussed the practicability of our method and the user's motivation to follow our recommendations. We show that our method still works in the case that not all users comply. In the next section, we will talk about another important issue in webs of trust systems. That is, how to differentiate reliable users from unreliable ones.

5.4 Differentiating reliable users from unreliable ones

In the previous sections, we have introduced how certificate recommendations may be used to verify the correctness of a name-to-key binding. In this section, we will focus on the issue of detecting unreliable users. The ability of detecting unreliable users is very important in webs of trust systems as many applications require to differentiate reliable users from unreliable ones. For example, on-line auction systems like eBay needs the reliable information of all participants for the purpose of safe selling and buying. In P2P systems, the reliability on users can enable us to determine the truthfulness of the information they provided. Basically, as human activity plays an essential role in security system, the reliability on them becomes very important. Our method of unreliable user detection is motivated by these observations and its intuition is explained in the following paragraph.

A simple fact regarding an incorrect name-to-key binding is that it must be certified by a unreliable user. Certificates are connected with each other and they form certificate chains. If we know a name-to-key binding is incorrect, then we may be able to trace back to the unreliable user who certify it. But it may be very hard to determine that a name-to-key binding is incorrect because only the owner of the name-to-key binding can declare its incorrectness and we have to trust what they said. This requirement makes it very hard to tell that a name-to-key binding is incorrect. However, it is very easy to tell conflicts between two name-to-key bindings. In the following, we suggest to make use of *certificate conflicts* and we will show how it may be used to detect unreliable users.

Definition 6 *A certificate conflict is a conflict between a pair of certified name-to-key bindings which disagree about a user's public key.*

Basically certificate conflict means two certificates report different public keys for a user. For example, $\langle Bob, k_1, s_{k_3}\rangle$ and $\langle Bob, k_2, s_{k_4}\rangle$ is a pair of conflicting certificates. The definition of certificate conflicts can be extended to the case that a user have multiple public keys by associating each public key with an index number. For example, Bob can have two public keys k_1 with index number 1 and k_2 with index number 2. $\langle Bob/1, k_1, s_{k_4}\rangle$ and

$\langle Bob/2, k_2, s_{k_5}\rangle$ don't create a conflict but $\langle Bob/1, k_1, s_{k_6}\rangle$ and $\langle Bob/1, k_3, s_{k_7}\rangle$ do. For simplicity, we omit the case of multiple keys and assume each user has only one key. The presented method in this section applies in the same way for both cases. Certificate conflicts are made by several reasons and we list them as follows.

1. An incorrect name-to-key binding and a correct name-to-key binding. In this case, the certifier of the incorrect name-to-key binding is unreliable.

2. Two incorrect name-to-key bindings. In this case, the certifiers of these two name-to-key bindings are both unreliable.

3. Two correct name-to-key bindings. This case indeed may happen if a unreliable user purposely does it. For example, John may present $(John, k_1)$ to Bob when Bob issues her a certificate and present $(John, k_2)$ to Alice when Alice issues her another certificate. These two name-to-key bindings create a conflict, but they are both correct. In this situation, user John is considered to be unreliable because he intentionally created the conflict while he had the choice of not creating it. It may not seem to be very beneficial for unreliable user John in doing so. But we may not simply omit this possibility because if we do, then either Bob or Alice will be framed as unreliable.

We have shown the three cases for certificate conflicts to happen. But in practice, it may not be easy to find out what the exact case is. Thus all the three possibilities have to be considered. And because of this, it is possible that we may mistake some good users as unreliable. To compromise this issue, we introduce the concept of *suspect set* for the purpose of detecting unreliable users.

Definition 7 *A suspect set is a set of identities, some of which belong to unreliable users. A member of the suspect set is called a suspect.*

The meaning of suspect is straight-forward here. Anyone who is possible to be a unreliable user is considered a suspect. If a user can't be excluded from being a suspect, then it is deemed as a suspect. In the following, we will show some intuitions for constructing suspects.

Suppose we have a certificate chain composed of four name-to-key bindings (Bob, k_1) $\rightarrow (John, k_2) \rightarrow (David, k_3) \rightarrow (Alice, k_4)$. What we know is that (Bob, k_1) is a correct

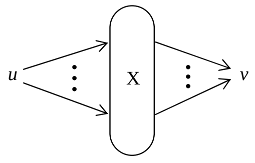


Figure 5.4: Suspect exclusion

name-to-key binding and $(Alice, k_4)$ is an incorrect one but don't know the correctness about the rest two. In this case, there are several possibilities. Both $(John, k_2)$ and $(David, k_3)$ may be correct and David is the unreliable user who certify the incorrect name-to-key binding $(Alice, k_4)$. It is also possible that only $(John, k_2)$ is correct and John certified the incorrect name-to-key binding $(David, k_3)$ and use the private key corresponding to k_3 to certify the incorrect name-to-key binding $(Alice, k_4)$. Another possibility is that the both of the two name-to-key bindings $(John, k_2)$ and $(David, k_3)$ are incorrect and Bob is the unreliable user who certified all the three incorrect name-to-key bindings. It is wrong to include only David or John or both in the suspect set because all the three users Bob and John and David are possible to be unreliable. The example shows some basic insights into constructing suspect set. To summarize, if a user can't be excluded from the possibility of certifying an incorrect name-to-key binding, then it is deemed as a suspect. Theorem 5 is used to determine if a user is a suspect.

Theorem 5 *A user is a suspect if and only if there exists a certificate chain starting from a certificate issued by the user and ending at a certificate which contains an incorrect name-to-key binding, and no certificate in the chain is known to contain a correct name-to-key binding.*

Now we prove Theorem 5. The "if" part of Theorem 5, is very straight-forward. In this case, it is possible that all the name-to-key bindings in the path are incorrect and are all certified by the unreliable users. If so, the user is also unreliable and has to be deemed as a suspect.

For the "only if" part of Theorem 5, we'd like to use the equivalent contra positive to prove it. That is, we will need to prove that if there is a correct name-to-key binding in each path connecting the user to the incorrect one, then the user is not a suspect. Figure 5.4 is used to illustrate this case.

In figure 5.4, vertex u represents the name-to-key binding of the user we are interested in and v represents the incorrect name-to-key binding. Because there exists a correct name-to-key binding in each path from u to v, we will be able to find a vertex separator from u to v which contains all correct name-to-key bindings. A vertex separator from u to v is a set of vertexes whose deletion will disconnect all paths from u to v. Set X represents such a vertex separator from u to v. Now we need to prove that the user of u can be excluded from being unreliable, i.e., it is not possible that the incorrect name-to-key binding v is certified by the user u. It is trivial to prove this by contradiction. Suppose the incorrect name-to-key binding is certified by the user of u, there must be a path which connects u to v and where no correct name-to-key binding exist. This violates the given condition that from u to v, there exists a vertex separator X which contains all correct name-to-key bindings. □

The suspect set construction algorithm that will be presented in this work is substantially different from the one presented in chapter 4. In chapter 4, the suspect sets constructed merely use the certificate conflict information to trace all possible unreliable users that may be responsible. The number of suspect sets can be very large as a suspect set will be constructed for each pair of conflicting certificates. The size of a suspect set constructed can be substantially large if most name-to-key bindings in the corresponding certificate chains can't be proved to be correct. The suspect set construction in this chapter is different. It is aggregate process that constructs a single suspect set based on all conflicting certificate information. Also in the process of constructing the suspect set, it will utilize the random certificate recommendation method to verify the correctness of some name-to-key bindings. With this information, it can more accurately locate the unreliable users and produce much a smaller suspect set.

Now we describe the algorithm for constructing suspect set. The construction of suspect set is an aggregate process for detecting each individual suspect. Clearly by Theorem 5, it can be done by backtracking starting from each incorrect name-to-key binding. In the backtracking, each user of the name-to-key binding encountered is collected as a suspect until it reaches a correct one. Because it is hard to determine that a name-to-key binding is incorrect, we will use certificate conflicts for constructing suspect sets. And we need to consider all the three possibilities of certificate conflicts in constructing the suspect set. The following algorithm 5 describes the process of constructing suspect set based on certificate conflicts. To make the process straight-forward, we add a virtual vertex s to the certificate graph and add an directed edge from each conflicting name-to-key binding to it. The backtrack starts from s and will proceed on all paths that terminate at s in the certificate graph. Algorithm 5 is used to construct the suspect set. It is actually a

Algorithm 5 Suspect Set Construction

input: Certificate graph G, virtual vertex s output: Suspect set S

```
1: $Q = \emptyset$
2: ENQUEUE($Q, s$)
3: **while** $Q \neq \emptyset$ **do**
4:    $u \leftarrow DEQUEUE(Q)$
5:    **for** each $v \in Predecessor[u]$ **do**
6:       $S = S \bigcup v$
7:       **if** $color[v]$=GRAY **then**
8:          $R = R \bigcup v$
9:       **end if**
10:   **end for**
11:   **if** $Q = \emptyset$ **then**
12:      **for** each $w \in R$ **do**
13:         **if** $color[w]$=VERIFY($w$)=DARKGRAY **then**
14:            ENQUEUE(Q,w)
15:         **end if**
16:      **end for**
17:      $R = \emptyset$
18:   **end if**
19: **end while**
```

modified BFS. In the running process of the algorithm, the vertexes in the certificate graph are painted with three colors. A vertex with WHITE color represents a correct name-to-key binding. A vertex with GRAY color represents a name-to-key binding that has not been visited and is unknown to be correct or not. A vertex with DARKGRAY color represents a name-to-key binding that has been visited and is unknown to be correct or not. Q is a first-in, first-out queue which stores all the vertexes to be examined in the backtracking. S is the suspect set. R is the set of name-to-key bindings whose correctness that we try to verify in the process of constructing suspect set. $Predecessor[u]$ is the predecessor vertex of u. A predecessor vertex of u is defined to be a vertex who has an directed edge that terminates at u. The function VERIFY(w) is to verify the correctness of the name-to-key binding represented by vertex w. The function returns either WHITE(correct) or DARKGRAY(unknown to be correct or not).

In general, the algorithm tries to search through the entire certificate graph by backtracking in each path that terminates at each conflicting name-to-key binding. In the backtracking, once it encounters a name-to-key binding that is unknown to be correct or not, it would try to verify its correctness using the VERIFY() function. The VERIFY() function can be any of the trust computation methods, including those in [54, 52, 59, 47, 12, 45, 42, 53, 36, 35] and our recommendation method presented in this paper. If it turns out to be correct, the backtracking will stop in that particular path. In the backtracking, all the identities of the name-to-key bindings we have encountered are collected as suspects. The output of the algorithm is a suspect set S. Now we explain the details of algorithm 5.

Line 1 and 2 initialize Q to the queue just containing the virtual vertex s. The **while** loop of lines 3-19 iterates as long as there remain gray vertexes, which are discovered vertexes that have not yet had their predecessors fully examined. Line 4 determines the dark gray vertex u at the head of the queue Q and removes it from Q. The **for** loop of lines 5-10 considers each vertex v which is the predecessor of u. Each vertex v we visited here is added to the suspect set S in line 6. If vertex v is gray, then it is also added to the verification set R in line 8 for verification in line 13. When Q becomes empty at line 11, it means that all predecessors of vertexes in Q have been examined and those gray vertexes have been put into set R for verifications. The **for** loop of lines 12-16 try to verify the correctness of each name-to-key bindings in the verification set R. If the name-to-key binding turns out to be correct after recommendation, then the vertex is painted with white color. Otherwise, it is painted with dark gray color and is enqueued into Q because we don't know if it is a correct name-to-key binding or not and we need to further examine its predecessors. Line 17 empty the verification set R because the tasks of verifying the correctness of name-to-key bindings in it is finished.

The suspect set construction algorithm can be run by any user who wants to detect malicious users. And because each user may have different knowledge about the truthfulness of certificates, the results they computed may be different.

It is worthwhile to point out the role of VERIFY() function in this algorithm. Verifications are made for a name-to-key binding that is unknown to be correct or not once we encounter it. If the name-to-key binding turns out to be correct after verification, then

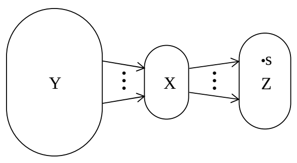


Figure 5.5: Separating trusted users from untrusted ones

the algorithm doesn't enqueue it to Q in line 14 and its predecessors won't be examined in the for loop of lines 5-10. It means all of its predecessors may not appear in the suspect set S in line 6. It is easy to see that the verification helps us reduce to size of suspect set. And of course, the smaller the suspect set, the better. The suspect set construction method is orthogonal to the random recommendation method presented in section 5.3. The VERIFY() function can be the certificate recommendation method in section 5.3 or any other method that can verify the correctness of the given name-to-key binding. We are aware of the fact that a verification may not always return a result. In this case, our method will still work but the resulting suspect set may be larger.

The goal of algorithm 5 is to construct a suspect set that includes all unreliable user's identities, as stated in Theorem 6.

Theorem 6 *The suspect set S constructed by algorithm 5 contains all unreliable user's identities and is optimal.*

By optimal, we mean that the suspect set we construct is minimal, i.e., no users ca be excluded from the suspect set. Now we prove Theorem 6. First, we prove that the suspect set constructed by algorithm 5 includes all unreliable user's identities. Figure 5.5 partitions the vertexes in the certificate graph into three sets. Set X contains all the white vertexes encountered in the backtracking. Set Z contains all the vertexes visited during the backtracking except those vertexes in X and clearly doesn't contain any white vertexes. Set Y contains the rest vertexes. $\Pi = X \bigcup Y \bigcup Z$. It is trivial to see that X is a vertex separator from Y to Z. That is, the deletion of X would disconnect every path from any vertex in Y to any vertex in Z. If there is any path from a vertex in Y to a vertex in Z after deletion of X, the backtracking should continue on this path until it meets a white vertex in Y, which is contrary to the condition that all such white vertexes are in X. Because each unreliable user has to appear in at least one certificate chain which starts from a correct name-to-key binding(represented by vertexes in X) and terminates at the incorrect name-

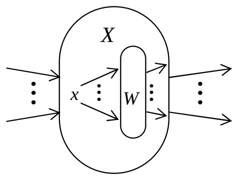


Figure 5.6: Proof of optimal suspect set construction

to-key binding(represented by vertexes in Z), all unreliable users' identities must appear in the set $X \bigcup Z$, which is exactly the suspect set that algorithm 5 constructs.

Second, we prove that the suspect set construct by algorithm 5 is optimal, i.e., no users in the constructed suspect set can be excluded. We'd like to use contradiction to prove it. Suppose that there is a suspect x that should be excluded from the suspect set. By Theorem 5, there must be a vertex separator W which contains all white vertexes from x to the virtual vertex s. Because there is no white vertexes in set Z, the vertex separator W must be in set X, as well as the vertex x. Figure 5.6 is used to describe this case.

It is easy to see that there is a contradiction for x to appear in set X because the backtracking process starting from s is not going to reach x due to the vertex separator W between x and s. $\square$

We have proved that the algorithm is correct and optimal in constructing the suspect set, now we discuss some other aspects of it. The algorithm is guaranteed to terminate and produce a solution. In the worst case, it will terminate after searching through the entire certificate graph and end up with a suspect set which includes all user's identities. For the running time, despite the VERIFY() function which is actually a human interaction process, the overall running time for the algorithm is the same as a BFS, which is $O(V+E)$. It is interesting to find out if the suspect set produced by the algorithm has a bound or not. If there is a bound, then the closer the bound to the actual number of unreliable users, the better. Assuming that all users would comply with the recommendations, it is easy to see that the constructed suspect set will include all the unreliable users and all the victims in the certificate graph. Another interesting property is that this suspect set construction method produces similar result as the one presented in section 3.4.3. In section 3.4.3, we present several rules to construct many different suspect sets. If we make a union of all the suspect sets constructs, it should equal to the suspect set constructed in this section, assuming that we have the same knowledge about the truthfulness of all certificates.

We have shown the methods to construct suspect set and we proved it to be correct and optimal. To see the actual performance of the method, we ran some experiments on real-world PGP keyrings. We implemented algorithm 5 and tested it to investigate its practicality and benefits. The best examples of webs of trust that are widely available are PGP keyrings[18]. We therefore used these for purposes of experimental validation. The PGP keyrings used in this experiment are downloaded from many PGP keyservers, e.g., pgp.mit.edu etc. The typical size of PGP keyring file is around 27,000 keys[58]. The keyanalyze [58] tool was used to extract many strongly-connected components. Strongly-connected components of PGP keyrings usually represent groups of users that were closely connected to each other and are good candidates for experiments. In these strongly-connected components, the size of each keyring range from 20 to 16449 keys.

In this experiment, we emulated the attacks as follows. First, we randomly select m unreliable users and n victims in the certificate graph. The m unreliable users colluded to issue m certificates which contains the same incorrect name-to-key binding for each victim, respectively. These $m \cdot n$ certificates were added to the certificate graph. These certificates which contain the incorrect name-to-key bindings create conflicts with the correct name-to-key bindings in the certificate graph. Second, we ran algorithm 5 on the certificate graph which contains these certificate conflicts. The VERIFY() function we used in this experiment is the certificate recommendation method that we proposed in this paper and is emulated as follows. Each user in the certificate graph was assigned a random generated compliance factor κ. The distribution of all users' κ is uniformly distributed between 0 and 1. Whenever a user received a request of issuing a certificate from the certificate recommendation, the user would generate a random number Υ between 0 and 1. If and only if Υ is less than κ, the user would follow the recommendation. If the number of users who follow the certificate requests and certify the requested name-to-key bindings is greater than α, then the VERIFY() function will return a true value to indicate that the name-to-key binding is correct. We ran the experiment by varying the value of β. As a result, the size of the suspect set will vary. And the smaller the $|S|$, the better.

Figure 5.7 and 5.8 show the results of our experiments of algorithm 5 on a PGP keyring with 1226 keys and 8780 certificates. The X axis represents β, i.e., the number of certificate requests in a recommendation. The Y axis represents the size of the constructed suspect set. Different curves represent results corresponds to different number of unreliable users and victims. In this experiment, wet set $\tau = 10\%$ and choose m to be 10, 50, 100 and

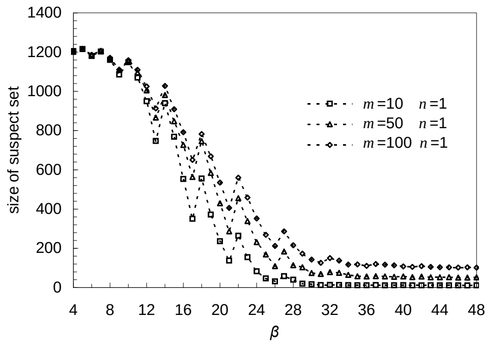


Figure 5.7: The size of suspect sets constructed by algorithm 5.

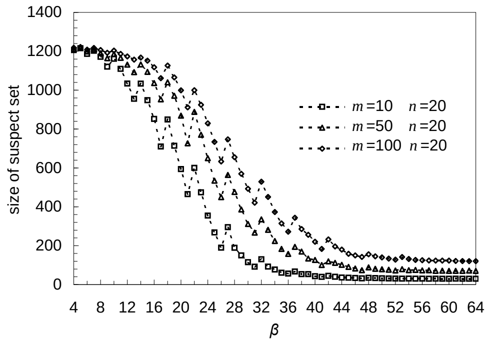


Figure 5.8: The size of suspect sets constructed by algorithm 5.

m,n	10, 1	50, 1	100, 1	10, 20,	50, 20	100, 20
lower bound of suspect set size	11	51	101	30	70	120
β	38	46	48	52	58	62
α	13	14	15	15	16	17

Table 5.1: minimal size of suspect sets constructed

n to be 1, 20. We believe that these numbers are representative and practical for showing how the suspect construction method works.

From figure 5.8, It is not surprising to see that roughly with the increase of β, the size of the suspect set decreases. But it also appears that in many small segments the size of the suspect set increases when β increases. The reason of these temporary drops is due to the rounding of the number α. For example, for the curve with $m = 10$ and $n = 1$, $\alpha = 7$ when $\beta = 13$. When β increases to 14, α increases and is rounded to 8. Clearly, the case of $\alpha = 7$ and $\beta = 13$ requires less percentage of user compliance than that of the case of $\alpha = 8$ and $\beta = 14$. Because user's compliance follows the same random pattern in these two cases, the number of name-to-key bindings that can be proved to be correct in the latter case would be lower than that in the earlier case. Thus the size of suspect set produced in the latter case is larger than that in the earlier case.

It appears that the decreasing rate of the size of the suspect set slows down as β increases and tends to stop after β reaches a certain point. It is because algorithm 5 is reaching the minimal suspect set by increasing β. The lower bound for the size of the suspect set is simply $m + n$. Algorithm 5 is able to achieve this lower bound when the $VERIFY()$ function returns true each time it is called. From the results of all experiments that we have ran, we found that this lower bound can be achieved with a reasonable number of β. These numbers are shown in table 5.1.

Table 5.1 shows the lower bound for the suspect set and the corresponding β and α. The first line represents the number of unreliable users and victims in the experiment, respectively. The second line represents the lower bound of suspect set size. The third line represents the smallest number of β that achieves the lower bound of the suspect size. And the fourth line represents the number of α corresponding to β. The tables shows that our suspect construction method is able to produce a minimal suspect set when β reaches a certain number. Though β seems to be a big number, α looks to be very reasonable. Experiments results have shown that these numbers are doable in the situation that users response to certificate requests in a random fashion.

All experiments were performed on a Pentium IV, 2.0GHZ PC with 512MB memory in Fedora Core 2 platform. Running times for any single experiment ranged from 3 to 30 seconds. We believe the running time is very efficient.

We have shown how to use certificate conflicts to detect unreliable users and present the algorithm to construct suspect set. The information from constructing suspect set is very useful because it helps us to differentiate reliable users from unreliable ones. In the next section, we will conclude our work in this paper and present some future work.

5.5 Conclusions

In this work we described how a distributed web of trust can be made more robust against unreliable attacks using certificate recommendations. The problem is modeled as a probability problem, i.e. decreasing the chances of unreliable behaviors by randomly selecting the certificate issuers. Our recommendation method for this problem is very straightforward and probability analysis has shown that even in the face of a large percentage of unreliable users, the PGP keyrings can be made almost 100% robust with only a small number of additional certificates to be issued. This method is novel because it proves it can deal with a very large percentage of unreliable users in very large networks with almost 100% guarantee on the correctness of name-to-key bindings. In addition, we addressed the issue of user non-compliance and show how to make redundant recommendations to overcome it. These recommendations methods can provide the recommendation service in many ways. For example, they can be coded in a piece of software and be downloaded and executed in the user's own computer. Of course in this case, this software has to be trusted to make these recommendations. It can also be provided by means of a trusted online service to eliminate the chances that the downloaded software may be compromised. In addition to the recommendations method presented in this paper, we also present the method of suspect construction and show how it can be used to differentiate reliable users from unreliable ones.

Chapter 6

Webs of trust in MANET

6.1 Introduction

Mobile ad-hoc networks(MANET) [51] are networks that are composed of wireless mobile devices. These devices communicate with each other via wireless channel links. MANET usually doesnt have network infrastructure support. There are no dedicated routers in the network. All network packets are forwarded by the nodes themselves. MANET has the following properties that make it very unique. First, nodes in MANET may move frequently around the network. The network topology will change with the movement of the network nodes. Nodes may have hard times communicating with each other because the routing has to be frequently updated if the network topology changes frequently. Second, all nodes in MANET connect with each other via wireless channel links. These wireless links have limited range and are frequently subject to wireless channel errors and collisions. Node disconnections may frequently happen due to these disruptions. Third, MANET nodes may have limited power, storage and bandwidth. Most MANET nodes are battery-powered and may last only a few hours. Especially for the wireless portable devices, such as PDA. They usually have very limited storage capacity and can't afford to high communication overhead. Last, MANET usually operates in severe situations, such as military operations and rescue missions. This make it more difficult for nodes to communicate with each other in MANET.

Regardless of these severe conditions, MANET is a very flexible network because there is no infrastructure needed. The network can be setup almost at the same time when the nodes are available. The nodes and the network can move freely from one location to another. This is particularly important for emergency situations, rescue missions and military operations where the network is required to be ready in a short time.

In this work, we would still like to work on the problem public-key authentication, in the context of mobile ad-hoc networks. In the wired network with infrastructures, public-key certificates are usually stored in a central certificate server, where users can retrieve them when needed. In mobile ad-hoc networks, there is no such central servers. We need to find an efficient way for users to be able to authenticate each other's public key.

[15] presents a self-organized approach for the problem of public-key authentication in MANET. In this self-organized approach, each node accumulates a subset of all certificates and store them at its local certificate repository. When two nodes want to authenticate each other's public key, they will merge their certificate repository and try to find a certificate chain between them.

In this work, we will still like to follow the self-organized approach because we believe that it is very suitable for MANET. The organization of this paper is as follows. Section 6.1 explains the basics of MANET and points out the problem of public-key authentication in mobile ad-hoc networks. Section 6.2 defines the research problem in this work. Section 6.3 presents the solutions to the research problem. Section 6.4 shows the performance of our solution on PGP keyrings and does comparisons with an existing solution. Section 6.5 analyzes the optimal solution, computes the lower bound and compares it with out solution. Section 6.6 presents another version of the solution which is suitable for the situation that communication problem happens. Section 6.7 discusses the case of network partitions and certificate issuance and revocation. The last section 6.8 concludes this work.

6.2 Problem Descriptions

In this section, we will present the problem definitions and the notations used in this work. In the network, a certificate is represented by $\langle X, k_x, s(k_w)\rangle$, where X is the identity, k_x is the public key that is bound to X, and $s(k_w)$ is the digital signature over the binding of X and k_x, which can be verified by public key k_w. Assume in the ad-hoc network, when user X issues user Y a public key certificate, it will keep a copy of the certificate and send a copy of the certificate to Y. Both X and Y will have this certificate locally. The set of all certificates is represented by R. Denote G the certificate graph that represents the set R of certificates. In the certificate graph, each vertex represents a public key and each edge represents a certificate. Each edge is labeled with the identity in the certificate. IIf there is a certificate $\langle X, k_x, s(k_w)\rangle$, there is an edge from k_w to k_x labeled with X. For simplicity, assume in the network each node has a single identity and a single public key. Thus, each vertex in the certificate graph represents a public key and corresponds to a node in the network. Each node x may accumulate a set R^x of certificates about other nodes. Denote G^x as the certificate subgraph that represents the set R^x of certificates and E^x the set of edges of G^x. How to obtain these certificates is the research problem to be solved in this work. Denote G^{xy} the certificate graph that represents the set $R^x \bigcup R^y$ of certificates and E^{xy} the set of edges of G^{xy}. Denote k_x x's public key.

The basic research problem in this work is to enable users to authenticate each other's public keys in MANET. The way that it works is to self-organized and fully distributed. As in [15], each node in the network will first collect a set of certificates from other users. When two nodes met each other, they will merge the set of certificates they collected and try to find a certificate chain between them. Denote $\mathcal{A}$ the algorithm used to construct each node X's local certificate repository R^x whose size is λ. For convenience, we will use E^x to represent R^x in the definition, $|E^x| = \lambda$.

The performance κ of the algorithm $\mathcal{A}$ to construct E^x is defined to be ratio of the number of pairs of (k_y, k_z) for which there exists a directed path from k_y to k_z in G^{yz} and the number of pairs of (k_y, k_z) for which there exists a directed path from k_y to k_z in G,

$$\kappa(\mathcal{A}, \lambda, G) = \frac{|\{(k_x, k_y) \in V \times V : k_x \rightsquigarrow k_y \exists\ G^{xy}\}|}{|\{(k_x, k_y) \in V \times V : k_x \rightsquigarrow k_y \exists\ G\}|}.$$

In this performance definition, $(k_x, k_y) \in V \times V$ means any pair of vertexes in G. $k_x \rightsquigarrow k_y \exists\ G^{xy}$ means there exists a path from k_x to k_y in G^{xy}. With this performance definition, we can formally define the problem to be solved in this work.

Problem 1 *Given a certificate graph G the size λ of each node's local certificate repository, design an algorithm $\mathcal{A}$ to construct each node X's subgraph G^x in a way such that $\kappa(\mathcal{A}, s, G)$ is maximized.*

Problem 1 explains the goal of this work is to enable all nodes to be able to authenticate each other in the MANET. The factor to be optimized is the size of each node X's local certificate repository R^x. Minimizing λ is advantageous not only in terms of storage but also in the aspect of communication overhead. The reason is that the certificates in each node's local certificate repository are delivered from other users, minimizing the number of delivered certificates can of course save the communication cost thereof.

We have defined the problem. Next we will present the solution to this problem.

6.3 Our solution

6.3.1 Basic approach

Our basic approach is motivated by the method in [15]. We have already explained the details of this approach in section 2.8. Generally say, each node will collect a set of certificates from other users by sending queries. When node X wants to authenticate node Y's public key, X will ask Y to send all the certificates she has collected to X, then X will try to find a certificate chain from itself to Y. The algorithm used for collecting certificates from other users is the maximum degree algorithm(MDA).

Motivated by this approach, we have designed an improved version of the solution. The intuition of our solution is we make use of certificate path information which makes our certificate collection more efficient. Our idea of making use of certificate path information comes from the distance vector routing protocols that we will explain in the next section.

6.3.2 Distance Vector Routing Protocols

The routing problem shares a lot of similarities with the problem of finding a path between two vertexes in the certificate graph. The certificate graph is like the network topology. A vertex in the certificate graph corresponds to a router in the network and an edge in the certificate graph corresponds to a link between routers. In our problem, we are trying to find a path from a vertex to another. In the routing problem, the goal is to find a path from a router to another router. Because of these similarities, we found it is a pretty good idea to make use of routing protocols for the purpose of exchanging certificate path information.

We first explains some basics of routing protocols. The routing protocols are classified into two categories, the link state protocols and the distance vector protocols[40]. Link state protocols usually require flooding the network with route information. Because flooding is expensive, we are not interested in using it in mobile ad-hoc networks because

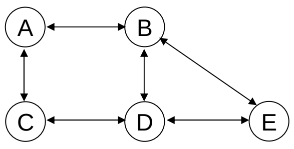


Figure 6.1: Distance vector routing protocols

destination	weight	line
A	0	-
B	1	B
C	1	C
D	∞	-
E	∞	-

Table 6.1: Node A's initial routing table

it will consume too much network bandwidth. Instead, we believe distance vector routing protocols are better choices because they only exchange routing information between local neighbors. In general, distance vector routing protocols work in the following way. Each node will maintain a routing table, which contains the best route to reach each other node in the network. The nodes periodically exchange and update routing information with their neighbor nodes. After a "convergence time", each node's routing table should reflect the best paths to reach all other nodes. The example in figure 6.1 illustrates how distance vector routing protocols work.

Assume the weight of each link in figure 6.1 is 1. Table 6.1 and 6.2 is A and B's initial routing table, respectively. In the routing table, the destination is the network node that the node needs to reach; the weight is the aggregated distance to reach that node; the line is the next hop to reach that node. Initially, each node in the network only knows the distance to the nodes that are directly linked to it. The routing table only contains routing information about its neighbor nodes. For example, initially, in routing table 6.1, node A only knows the routing information for B and C, but has no idea of how to reach D and E.[1]

To get the information on how to reach all the network nodes, each node will periodically exchange their routing tables with its direct neighbor nodes and update its routing table. Table 6.3 shows the A's updated routing table after it has exchanged its

[1]In actual routing protocols, there is always a default route which can be used for destinations that the node doesn't know how to reach.

destination	weight	line
A	1	A
B	0	-
C	∞	-
D	1	D
E	1	E

Table 6.2: Node B's initial routing table

destination	weight	line
A	0	-
B	1	B
C	1	C
D	2	B
E	2	B

Table 6.3: Node A's routing table after exchanging with B

routing table with B.

In table 6.3, A updates its entries for D and E. This is because after A exchange routing tables with B, A knows that the distance from B to both D and E is 1. Since A's distance to B is only 1, then A's distance to both D and E is only 2 through B, which is smaller than its original distance to them in table 6.1. Thus A updates its distance to both D and E to be 2 and the line to be B.

As the example shows, distance vector routing protocols try to find the shortest path between two network nodes by exchanging and updating routing tables between neighbor nodes. After a certain amount of time, each node should have the shortest path information to reach all other network nodes. Distance vector routing protocols are always accused of not being scalable in networks with thousands of nodes. And they are also considered inefficient when routes change frequently. We believe that these will not be big problems for the certificate graph in the ad-hoc networks. First, thousands of nodes may be too small for the case of Internet, but we believe it is a sufficient large number for most ad-hoc networks. Second, certificate graph doesn't change very often. New certificates may be generated and some old certificates may be revoked. But the rate of change is much slower than that of the routing protocols.

destination	distance	next

Table 6.4: Format of node X's outgoing path table

There are real-world versions of distance vector routing protocols, such as RIPv1[31], which uses distributed Bellmann-Ford algorithm[11] for determining the best route. These routing protocols are also known for its "'counting to infinity"' and "'routing table loops"' problems[27], but there are known solutions for these problems, such as those in [28]. Because of the similarities of our problem and the routing problem, it is possible to directly apply the distance vector routing protocols in our solution, with only some semantic changes. We should also note that the routing protocols we have discussed in this section have nothing related to ad-hoc routing protocols. We intend to use the distance vector routing algorithm on the certificate graph, not the actual ad-hoc networks. The node's mobility in ad-hoc networks won't affect how our solution works because it doesn't change the certificate graph.

We have discussed the similarities between the distance vector routing protocols and finding certificate path. Now we turn to our solution. There are basically two components in our solution. One is the certificate path exchange and the other is certificate collection. In the certificate path exchange, nodes will exchange and construct the information on certificate paths to/from other nodes. In the certificate collection, each node will use these information and collect certificates from other users to construct its local certificate repository. In the next two sections, we will explain them in details.

6.3.3 Certificate Path Exchange(CPE)

The certificate path exchange works in the similar way to the distance vector routing protocols and is explained as follows. In the certificate path exchange, each node will maintain two tables to store the path information from its own vertex to other vertexes and from other vertexes to its own vertex in the certificate graph, respectively. The outgoing path table contains information about the shortest paths from its own vertex to other vertexes. The incoming path table contains information about the shortest paths from other vertexes to its own vertexes.

Table 6.4 and 6.5 show the entries of X's outgoing path table and incoming path table, respectively. In table 6.4, "destination" is the vertex to be reached from k_x; "distance"

source	distance	previous

Table 6.5: Format of node X's incoming path table

is the number of edges in the corresponding path; "next" is the next vertex on the path. In table 6.5, "source" is the vertex that can reach k_x; "distance" is the number of edges in the corresponding path; "prev" is the previous vertex of k_x on the path.

Initially when the network is setup, each node's outgoing path table only contains information about how to reach its direct neighbor vertexes from its own vertex in the certificate graph. The node has such information locally because the outgoing edges from its own vertex in the certificate graph represent the certificates that the node issues to others.

To get information about how to reach other nodes, each node will periodically send its outgoing path table to the nodes corresponding to the vertexes that are adjacent to its own vertex in the certificate graph. A node X is adjacent to node Y is there is an edge from Y to X. After the node receives the table, it will use these information to update its own outgoing path table. The algorithm used is the same as the one used in the distance vector routing protocols, i.e., the distributed Bellman Ford algorithm[11]. Simply say, it works in the following way. For each destination vertex, the node will compare the current distance with the corresponding distance in the received table. If the distance in the received table plus 1 is smaller than the current distance, the node will update the corresponding distance and the next vertex. If the two distances are the same, the node will compare the two corresponding next vertexes and choose the one with a higher out-degree. A vertex's out-degree equals to its number of outgoing edges.

Algorithm 6 is used to update the outgoing path table at each node. $distance(k_x)$ is the distance to reach k_x in the certificate graph. $next(k_x)$ is the next node to reach k_x. $outdegree(k_x)$ is the out-degree of k_x, i.e., the number of outgoing edges of vertex k_x in the certificate graph. We now explain algorithm 6. Lines 1 starts a loop to process each destination k_x in the received path table from node Y. Lines 2-4 first compare the current distance to reach k_x with the corresponding distance from Y plus 1. If the current distance is bigger, then it will update its distance to the shorter one. Lines 5-9 do the same comparison and if the distances are the same, it will compare the outgoing degree of the current next

Algorithm 6 The algorithm to update the outgoing path table

```
1: for each destination $k_x$ in the received path table $Y$ do
2:     if $distance(k_x) > Y.distance(k_x) + 1$ then
3:         $distance(k_x) = Y.distance(k_x) + 1$
4:         $next(k_x) = k_y$
5:     else if $distance(k_x) = Y.distance(k_x) + 1$ then
6:         if $outdegree(next(k_x)) > outdegree(Y.next(k_x))$ then
7:             $next(k_x) = k_y$
8:         end if
9:     end if
10: end for
```

vertex to on the path to k_x with the outgoing degree of Y's next vertex on the path to k_x, and choose the one whose outgoing degree is bigger. The intuition is straight-forward, if the two next vertex lead to k_x in the same distance, then choosing the one with the bigger outgoing degree will increase the chances to reach more vertexes in shorter path. Line 10 completes the loop process which starts at line 1.

The incoming path table is constructed and updated in a similar way, except that nodes are exchanging maintaining incoming path table information. At the very beginning of the network, each node's incoming path table only contains information about how other vertexes may reach its own vertex in the certificate graph. The node has such information locally because the incoming edges to its own vertex in the certificate graph represent the certificates other nodes issued to it.

To get information about how other nodes may find certificate paths to itself, each node will periodically send its incoming path table to the nodes corresponding to those vertexes that are adjacent from its own vertex in the certificate graph. A node X is adjacent from node Y is there is an edge from X to Y. After the node receives the table, it will use these information to update its own incoming path table. The algorithm used is similar to algorithm 6. Simply say, it works in the following way. For each source vertex, the node will compare the current distance with the corresponding distance in the received table. If the distance in the received table plus 1 is smaller than the current distance, the node will update the corresponding distance and the previous vertex. If the two distances are the same, the node will compare the two corresponding previous vertexes and choose

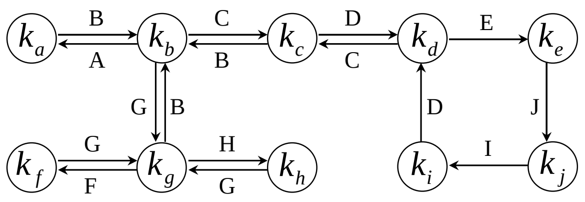


Figure 6.2: MSA example certificate graph

destination	distance	next
k_a	0	-
k_b	1	k_b

Table 6.6: Node A's initial outgoing path table

the one with a higher in-degree. A vertex's in-degree equals to its number of incoming edges. We don't present the algorithm to update each node's incoming path table as it is very similar to algorithm 6.

Next, we use the certificate graph in figure 6.2 as an example to show how a node's path tables are constructed and updated. Table 6.6 and 6.7 are node A and B's initial outgoing path tables, respectively. In these two initial path tables, they only have the knowledge of its directly connected outgoing neighbor vertexes in the certificate graph. After B sends its outgoing path table to A, node A's updated outgoing path table is shown in table 6.8. In 6.8, the entries for k_g and k_c are updated using the received outgoing path table 6.7 from B. Table 6.9 is node A's final outgoing path table after each node's path table information has been populated to all other nodes. In this table, node A has all the shortest path information from its own vertexes to other vertexes in the certificate graph. Table 6.10 is the incoming path table. It is different from A's outgoing path table. It contains the

destination	distance	next
k_a	1	k_a
k_b	0	-
k_g	1	k_g
k_c	1	k_c

Table 6.7: Node B's initial outgoing path table

destination	distance	next
k_a	1	k_a
k_b	0	-
k_g	2	k_b
k_c	2	k_b

Table 6.8: Node A's updated outgoing path table after receiving updates from B

destination	distance	next
A	0	-
k_b	1	k_b
k_c	2	k_b
k_d	3	k_b
k_e	4	k_b
k_f	3	k_b
k_g	4	k_b
k_h	3	k_b
k_i	6	k_b
k_j	5	k_b

Table 6.9: Node A's outgoing path table

shortest path information of how other vertexes may reach their own vertex.

Certificate path exchange requires network nodes to exchange their path tables periodically between nodes whose public keys are neighbors in the certificate graph. This adds communication overhead to the network. We measure the communication overhead by the extra communication bandwidth required by CPE. Because messages need to be transmitted over multiple hops in MANET, the number of hops between two communication nodes will also be taken into account. Assume the size of both incoming and outgoing path table is λ, the average number of hops between two communication nodes is h, and each node will exchange its path tables with other nodes in a frequency of f, the overall communication overhead for the network is $2\lambda \cdot h \cdot f \cdot |E|$, where E stands for the set of edges in the certificate graph. Assuming the maximum length of all pairs shortest path in the certificate graph is m, the convergence time for updating path tables is $m \cdot \tau$.

We have described how the certificate path exchange works in this section. In the next section, we will describe the second part of our solution, i.e., the Maximum Shortest

source	distance	previous node
k_a	0	-
k_b	1	k_b
k_c	2	k_b
k_d	3	k_b
k_e	4	k_b
k_f	3	k_b
k_g	4	k_b
k_h	3	k_b
k_i	4	k_b
k_j	5	k_b

Table 6.10: Node A's incoming path table

Path Algorithm that is used in the process of certificate collection.

6.3.4 Maximum Shortest-path Algorithm(MSA)

By using the certificate path exchange, each node is able to construct the certificate path tables that contain shortest path information from its own vertex to other vertexes and from other vertexes to its own vertexes in the certificate graph. With the outgoing and incoming path tables for each node, we now describe how the process of certificate collection works.

Each node will do a modified DFS(Depth-First-Search)[30] on the certificate graph. The process starts from the node's own public key in the certificate graph. The resulting DFS tree contains all the edges that represent the certificates that the nodes needs to collect. Standard DFS algorithms can be used with a trivial modification for this purpose. The difference is that in the modified DFS, each time the node needs to select a vertex, we don't select it sequentially like what is done in the standard DFS algorithm, but we select it using own criteria. For outbound collection, each time we need to select a vertex, we will select the one which appears most frequently in the next column in the current node's outgoing path table. Similarly for inbound collection, each time we will select the vertex which appears most frequently in the prev column in the current node's incoming path table. To prevent duplicate selection, a vertex that has already been selected couldn't be selected again. The constructed DFS tree represents those certificates that need to be collected. The Maximum Shortest-Path Algorithm(MSA) for outbound collection is presented in algorithm

7.

Algorithm 7 Maximum Shortest-Path Algorithm for outbound collection of node X

```
1: $S = \emptyset$, $count = 0$, $E^x_{out} = \emptyset$
2: $\forall k \in V, visited(k) = 0$
3: $S.push(k_x)$
4: $visited(k_x) = 1$
5: while $S \neq \emptyset$ && $count < \frac{\lambda}{2}$ do
6:   $k_y = S.top()$
7:   $k_z = k, \forall k' \in N_y$ && $visited(k') = 0, f(k) \geq f(k'), k \in N_y$ && $visited(k) = 0$
8:   if $k_z = \varnothing$ then
9:     $S.pop()$
10:  else
11:    $E^x_{out} = E^x_{out} \bigcup \{k_y \rightarrow k_z\}$
12:    $count = count + 1$
13:    $S.push(k_z)$
14:    $visited(k_z) = 1$
15:  end if
16: end while
```

We now explain algorithm 7. S is a last-in first-out stack. It is used to maintain the list of vertexes that have been visited and will be revisited if needed in the certificate graph during the DFS. *count* is a counter to make sure that the number of certificates we collect will not exceed $\frac{\lambda}{2}$. E^x_{out} contains the set of edges we collect in the outbound collection. The $visited(k)$ function marks k as a visited vertex. Line 1 first initialize S, *count* and E^x_{out}. Line 2 initialize all vertexes in V as unvisited. Line 3 pushes root vertex k_x of the DFS tree into the stack. Line 4 sets vertex k_x as visited. Lines 5 starts the while loop for DFS. The while loop ends because either there is no vertex to search($S = \emptyset$) or we have collected the defined number($\frac{\lambda}{2}$) of certificates. Line 6 pops out the top member k_y of S. Lines 7 examines all k_y's outgoing neighbor vertexes and select the one k_z has not been visited and appears most frequently as the next vertex in Y's outgoing path table N_y. The intuition is very straight-forward. If the vertex can lead us to most number of other vertexes via shortest path, then by selecting it, the chance of connecting to other vertexes will be higher. Line 8 examines if k_z exists or not. If not, in line 9, we pop out k_y from

	A	B	C	D	E	F	G	H	I	J
k_a	0,-	1,A	2,k_b	3,k_c	6,k_j	3,k_g	2,k_b	3,k_g	4,k_d	5,k_i
k_b	1,k_b	0,-	1,k_b	2,k_c	5,k_j	2,k_g	1,k_b	2,k_g	3,k_d	4,k_i
k_c	2,k_b	1,k_c	0,-	1,k_c	4,k_j	3,k_g	2,k_b	3,k_g	2,k_d	3,k_i
k_d	3,k_b	2,k_c	1,k_d	0,-	3,k_j	4,k_g	3,k_b	4,k_g	1,k_d	2,k_i
k_e	4,k_b	3,k_c	2,k_d	1,k_e	0,-	5,k_g	4,k_b	5,k_g	2,k_d	3,k_i
k_f	3,k_b	2,k_g	3,k_b	4,k_c	7,k_j	0,-	1,k_f	2,k_g	5,k_d	6,k_i
k_g	2,k_b	1,k_g	2,k_b	3,k_c	6,k_j	1,k_g	0,-	1,k_g	4,k_d	5,k_i
k_h	3,k_b	2,k_g	3,k_b	4,k_c	7,k_j	2,k_g	1,k_h	0,-	5,k_d	6,k_i
k_i	6,k_b	5,k_c	4,k_d	3,k_e	2,k_j	7,k_g	6,k_b	7,k_g	0,-	1,k_i
k_j	5,k_b	4,k_c	3,k_d	2,k_e	1,k_j	6,k_g	5,k_b	6,k_g	3,k_d	0,-

Table 6.11: Outgoing path table of all nodes

the stack because it will not be revisited. If yes, in line 11-14, we will collect the current edge, increase the counter, add k_z to S for future searching and set k_z as visited. Line 16 completes the while loop.

We should also note that performing the DFS on the certificate graph doesn't require us to have the knowledge of the complete certificate graph. Each time we need to select the next vertex, we can query the node corresponding to the current selected vertex about the vertex that appears most frequently in the next or prev column in its certificate path table. Only local knowledge is required.

The algorithm for inbound collection works in the same way, except that the path table used is the incoming path table, the vertex selected is the previous vertex and the set of collected edges is E^x_{in}. Each edge in the set E^x_{in} and E^x_{out} corresponds to certificate. The two set E^x_{in} and E^x_{out} of edges correspond to the two set R^x_{in} and R^x_{out} of certificates , respectively. X's local certificate $R^x = R^x_{in} \bigcup R^x_{out}$, $|R^x| = \lambda$.

We now present an example that illustrates how the Maximum Shortest-Path algorithm works. Table 6.11 and 6.12 are the outgoing and incoming certificate path tables for all nodes, respectively. In these two tables, each column represents a node's path table. The number and letter in each column entry represent the distance and the next or previous vertex, respectively. Suppose $\lambda = 6$, for node A's outbound collection, it will select k_b, k_c and k_d. $E^A_{out} = \{k_a \xrightarrow{B} k_b, k_b \xrightarrow{C} k_c, k_c \xrightarrow{D} k_d\}$. For node A's inbound collection, it will select vertex k_b, k_c and k_d. The set of certificates node A collected is $E^A_{in} = \{k_b \xrightarrow{A} k_a, k_c \xrightarrow{B} k_b, k_d \xrightarrow{C} k_c\}$. $E^A = E^A_{in} \bigcup E^A_{out}$. For node I's outbound collection, it will select

	A	B	C	D	E	F	G	H	I	J
k_a	0,-	1,k_a	2,k_b	3,k_c	4,k_d	3,k_g	2,k_b	3,k_g	6,k_j	5,k_e
k_b	1,k_b	0,-	1,k_b	2,k_c	3,k_d	2,k_g	1,k_b	2,k_g	5,k_j	4,k_e
k_c	2,k_b	1,k_c	0,-	1,k_c	2,k_d	3,k_g	2,k_b	3,k_g	4,k_j	3,k_e
k_d	3,k_b	2,k_c	1,k_d	0,-	1,k_d	4,k_g	3,k_b	4,k_g	3,k_j	2,k_e
k_e	6,k_b	5,k_c	4,k_d	3,k_i	0,-	7,k_g	6,k_b	7,k_g	2,k_j	1,k_e
k_f	3,k_b	2,k_g	3,k_b	4,k_c	5,k_d	0,-	1,k_f	2,k_g	7,k_j	6,k_e
k_g	2,k_b	1,k_g	2,k_b	3,k_c	4,k_d	1,k_g	0,-	1,k_g	6,k_j	5,k_e
k_h	3,k_b	2,k_g	3,k_b	4,k_c	5,k_d	2,k_g	1,k_h	0,-	7,k_j	6,k_e
k_i	4,k_b	3,k_c	2,k_d	1,k_i	2,k_d	5,k_g	4,k_b	5,k_g	0,-	3,k_e
k_j	5,k_b	4,k_c	3,k_d	2,k_i	3,k_d	6,k_g	5,k_b	6,k_g	1,k_j	0,-

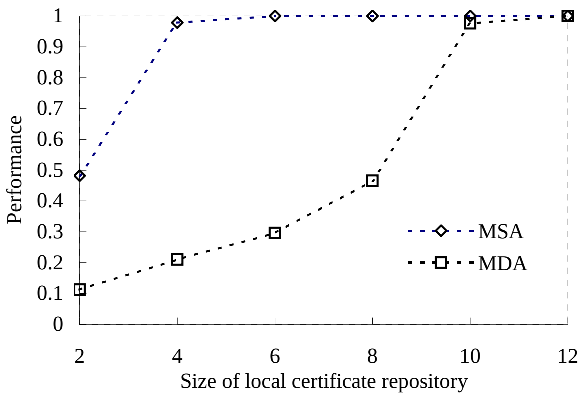
Table 6.12: Incoming path table of all nodes

vertex k_d, k_c and k_b. $E^I_{out} = \{k_i \xrightarrow{D} k_d, k_d \xrightarrow{C} k_c, k_c \xrightarrow{B} k_b\}$. For node I's inbound collection, it will select vertex k_j, k_e and k_d. $E^I_{in} = \{k_j \xrightarrow{J} k_i, k_e \xrightarrow{E} k_j, k_d \xrightarrow{D} k_e\}$. $E^I = E^I_{in} \bigcup E^I_{out}$. If we merge E^A and E^I, there is clearly a path between k_a and k_j in both directions.

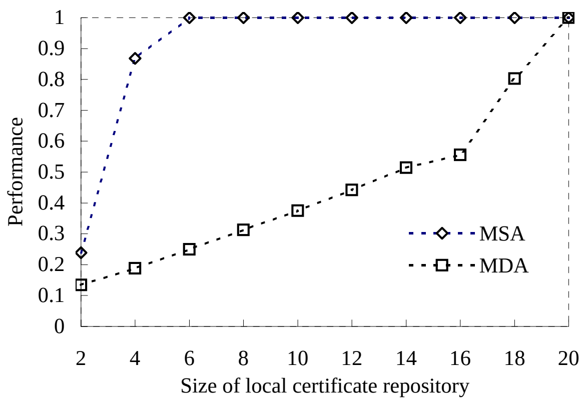
We have shown how our Maximum Shortest-Paths Algorithm works and analyze it. In the next section, we will do some experiments and show how our algorithm performs compared to others.

6.4 Experiment Results

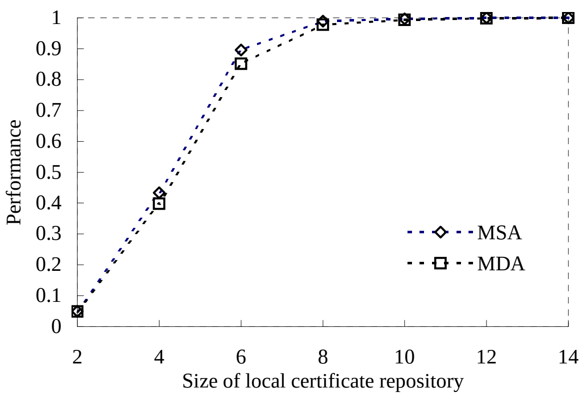
We did several experiments to test the performance of our algorithm. We also compared our algorithm's performance with the Maximum Degree Algorithm in [15]. The experiment data we used are PGP keyrings. These PGP keyrings were downloaded from keyservers, such as pgp.dtype.org etc. The total number of keys in these keyservers is in the amount of millions. We used the PGP keyanalyze[58] tool to extract many strong-connected components from the keyring files we downloaded from the keyserver. The strong-connected PGP keyrings we used in these experiments are labeled with their number of keys and number of certificates. For example, the keyring KR588-5237 represents the PGP keyring with 588 public keys and 5237 certificates. We did our experiments on 23 PGP keyrings, whose number of keys range from 21 to 1226 and whose number of certificates range from 47 to 8780. For all these PGP keyrings, experiment results show that our MSA



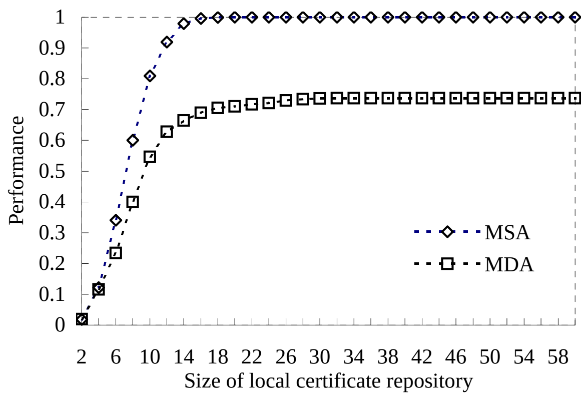
(a) KR34-451



(b) KR27-554



(c) KR588-5237



(d) KR1226-8815

Figure 6.3: Performance of MSA and MDA on some PGP keyrings

algorithm performs consistently better than the MDA algorithm. In all the cases, the MSA's performance converges to 100% very fast. But for MDA, in most cases, its performance converges very slowly to 100%. Note in all these experiments, we assume an ideal condition for the CPE mechanism to work. That is, each node's path tables will always contains the shortest path information. In the real-world environment of mobile ad-hoc networks, the communication condition can be very severe, and the CPE mechanism may not work 100%. This means that the path tables at each node may not always contain the best shortest path information, and the performance of MSA will be affected in this situation.

Figure 6.3 shows the performance comparison of the MSA and MDA. The X axis represents the size of each node's local certificate repository. In MDA, it is $2 \cdot \alpha \cdot \beta$ and in MSA, this is λ. For comparisons, we first set α and β, then let $\lambda = 2 \cdot \alpha \cdot \beta$. We did

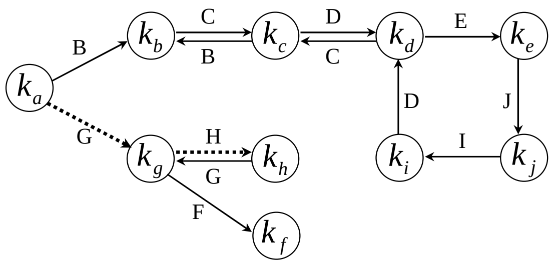


Figure 6.4: Example: MDA unable to collect required number of certificaes.

experiments for different values of α and found out that the MDA has its best performance when $\alpha = 1$. For all the experiments in this section, $\alpha = 1$. In figure 6.3, it is clear to see that the MSA algorithm performs consistently better than MDA. In figure 6.3(a), the MSA reaches 100% performance when the size of local certificate repository is 6. While the MDA reaches 100% performance when the size of local certificate repository is 12. The performance difference in figure 6.3(b) is much bigger. When the MSA reaches 100% performance, the MDA's performance is only 25%. The number of certificates that MDA needs to reach 100% performance is $\frac{20}{6} = 3.3$ times that of MSA. In figure 6.3(c), the performance of the two algorithms are very close to each other. MSA's performance is just slightly better than that of MDA. The reason is that most of the certificates collected are the same in the two algorithms. Specifically, in this particular certificate graph, most of the vertexes that can lead to most other vertexes via shortest paths are also the vertexes with maximum degrees. Since most of the selected vertexes in the MSA and MDA are the same, the performance of the two algorithms are very close to each other. In figure 6.3(d), we can see that MSA is very efficient. Its performance increases very fast as λ increases. At the point that $\lambda = 20$, its performance becomes 1. The MDA's performance seems to be quite unusual. After the performance reaches 0.7373 at $\lambda = 32$, it remains unchanged as λ increases to 60. We investigated this scenario and found it is a deficiency in the MDA. Given a α and β, in some cases, the certificate collection using MDA may not be able to collect $2 \cdot \alpha \cdot \beta$ number of certificates. When the value of $2 \cdot \alpha \cdot \beta$ is over the maximum number of certificates that MDA can collect, clearly the performance won't change any more. One scenario that MDA can't collect the required number of certificates is illustrated by the example in figure 6.4.

In figure 6.4, node A is trying to collect certificates in $\alpha = 1$ path with $beta = 3$.

First it will select vertex k_g, then vertex k_h. At this point, node A has collected two certificates but it can't go any further and it can't go back to k_g because k_g is already selected. This example illustrates one reason why MDA fails to collect the required number of certificates and explains the strange behavior of MDA in figure 6.3(d). Our certificate collection using MSA doesn't have this problem. It is based on DFS and will always search through all connected vertexes and be able to select the required number of certificates. In the case that λ is greater than the number of all connected vertexes, the method will return a set of certificates where we will be able to find a path to each connected public keys.

6.5 Optimality Analysis of MSA

In the previous section, we have shown that MSA performs much better than MDA. In this section, we are interested in finding out how close the MSA is to the optimal solution. Because the optimal solution is unknown at this time, it would be useful if we can compute a lower bound and compare with it.

To compute a lower bound, we look at the very basic way that public-key authentication works. The authentication can be done by finding a certificate path between two nodes. In the certificate graph, it corresponds to find a path between two vertexes. It is straight-forward to see that the number of certificates required will not be less than the length of the shortest path between the two vertexes. For example, if node X wants to authenticate node Y's public key, and the length of the shortest path from k_x to k_y is 4 in the certificate graph, the least number of certificates that X requires to authenticate k_y is 4.

In this process of collecting certificates, each node will perform both the outbound and inbound collection. Outbound collection is to collect certificates in the node's outgoing certificate paths. Inbound collection is to collect certificates in the nodes' incoming certificate paths. It is easy to see that if X's outbound collection and Y's inbound collection have an intersection(i.e., G^X_{out} and G^Y_{in} share a common vertex), then we would be able to find a certificate path from k_x to k_y in $G^X_{out} \bigcup G^Y_{in}$. If the shortest path from k_x to k_y has a length of 4, it means $|E^X_{out}| \geq 2$ and $|E^Y_{in}| \geq 2$ is the minimum requirement that we can find a path between k_x to k_y in $G^X_{out} \bigcup G^Y_{in}$.

Figure 6.5 is an example of node A's outbound certificate collection and node E's inbound collection. The dotted lines represent node A's outgoing path of certificate

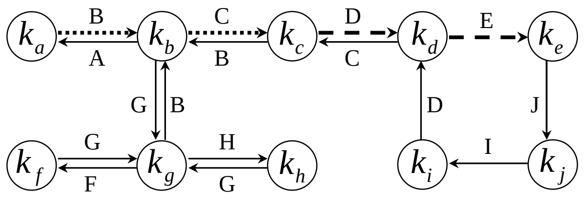


Figure 6.5: Outbound and inbound certificate collection

collection. The corresponding $E_{out}^{A} = \{k_a \xrightarrow{B} k_b, k_b \xrightarrow{C} k_c\}$. The dashed lines represent node E's incoming path of certificate collection. The corresponding $E_{in}^{E} = \{k_d \xrightarrow{E} k_e, k_c \xrightarrow{D} k_d\}$. G_{out}^{A} and G_{in}^{E} share a common vertex k_c in the certificate graph. In this example, they totally number of certificates both A and E collect is equal to 4, which is exactly the shortest path from k_a to k_e in the certificate graph.

Now we are able to compute the lower bound for the number of certificates that each node has to collect in order to find a certificate path between each. For any pair of node X and Y, the number of certificates X collects during its outbound collection, plus the number of certificates Y collects during its inbound collection, should be greater or equal to the shortest path length from k_x to k_y in the certificate graph. Let U be the set of all nodes. Let $\xi(X,Y)$ represent any path from k_x to k_y in the certificate graph. The length of the path from k_x to k_y is represented by $|\xi(X,Y)|$. Formally,

$$|R_{out}^{X}| + |R_{in}^{Y}| \geq \min(\xi(X,Y)), (X,Y) \in U \times U.$$

Because both R_{out}^{X} and R_{in}^{Y} are collected separately, and $|R_{out}^{X}| = |R_{in}^{Y}| = \frac{\lambda}{2}$, it's actually,

$$\frac{\lambda}{2} \geq \lceil \frac{\max(\min(\xi(X,Y)))}{2} \rceil, (X,Y) \in U \times U.$$

The lower bound of $\frac{\lambda}{2}$ we found is the integer roundup of the fraction of the maximum of the shortest path lengths of all pairs of vertexes and 2. In the next, we will do some experiments on real PGP keyrings and see how close our solution is to this lower bound.

Table 6.13 shows comparison between the lower bound and the value of $\frac{\lambda}{2}$ needed by our method to reach 100% performance for some PGP keyrings. We can see in most PGP keyrings, our method's $\frac{\lambda}{2}$ is exactly the same as the lower bound and for the rest cases,

keyring	KR105-243	KR67-194	KR25-208	KR26-129	KR26-232
lower bound	2	7	2	2	3
$\frac{\lambda}{2}$	2	8	3	2	4
keyring	KR27-554	KR27-66	KR27-77	KR28-169	KR28-122
lower bound	2	5	4	3	3
$\frac{\lambda}{2}$	3	6	4	4	3
keyring	KR31-145	KR32-142	KR33-133	KR34-451	KR67-192
lower bound	4	3	3	2	7
$\frac{\lambda}{2}$	5	3	3	3	8

Table 6.13: Comparison of lower bound and our solution

the difference is only 1. Clearly, the comparison shows that the performance of our solution is very close to the optimal one.

We should also note that in this section, the conditions of optimality analysis of our solution is subtly differently from the one in section 6.3. In section 6.3, when user X wants to authenticate Y's public key k_y, X will request Y to send $R^Y = R^Y_{in} \bigcup R^Y_{out}$ to itself. X will merge R^Y and R^X and try to find a certificate path from k_x to k_y in $G^Y \bigcup G^X$. As we have shown in this section, we know that the construction of R^X_{out} and R^Y_{in} is for the purpose of finding the path from k_x to k_y. While the construction of R^X_{in} and R^Y_{out} is for the purpose of finding the path from k_y to k_x. It would be unfair to compare $|R^X|+|R^Y|$ with the shortest path length from k_x to k_y. For this reason, we only compare $|R^X_{out}|+|R^Y_{in}|$ with the shortest path from k_x to k_y in this section. To be consistent, for all the experiments in this section, for any pairs (X, Y) of nodes, we didn't merge R^X and R^Y but only merge R^X_{out} and R^Y_{in} for the purpose of finding out if there's a path from k_x to k_y.

6.6 On-demand Certificate Path Discovery(OCPD)

Either the MSA or MDA is based on a probabilistic approach. That means both of them can't guarantee to find a certificate path between any two nodes. In this section, we will seek to a solution that can guarantee to find a certificate path between two nodes provided there is one.

In the CPE process, nodes exchange shortest path information. The function that a node needs to find a certificate path to another node is very similar to network packet routing in the wired network. In distance vector routing protocols, routers will use their

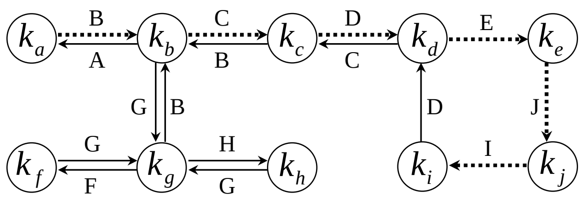


Figure 6.6: Certificate Path Discovery

routing tables to determine the best link to forward a packet. Similarly, the outgoing certificate path table at each node provides information on how a node can reach another in the certificate graph. By using this information, we are able to find a certificate path from one node to another. To find a certificate path from k_x to k_y in the certificate graph, we can do the following. First node X will look up its outgoing path table and find the next vertex k_1 to reach the destination vertex k_y. The edge from k_x to k_1 represents the first certificate on the path from k_x to k_y and X should have it locally because it is issued by X. Next, X will query the owner of k_1(suppose X_1) about its next vertex to reach k_y. X_1 will reply with the next vertex k_2 and the corresponding certificate. X will accept the certificate and continue to query X_2 about its next vertex to reach k_y. The process continues until we reach the vertex k_n that is directly connected to k_y in the certificate graph. When X query X_n about its next vertex to reach k_y, X_n will reply with a certificate that certifies k_y. This completes the certificate path discovery process and X now has a chain of $n+1$ certificates from k_x to k_y and will be able to authenticate k_y. Figure 6.6 illustrates how the certificate path discovery process works.

In figure 6.6, node A tries to discover a certificate path from k_a to k_i. By looking up its outgoing certificate path table, it knows that k_b is the next vertex on the path to k_i. Next, node A will query B about its next vertex to k_i. Node B will reply with k_c and the corresponding certificate $\langle C, k_c, s(k_b)\rangle$. Following the same process, node A will query node C, D, and E and they will reply with certificate $\langle D, k_d, s(k_c)\rangle$, $\langle E, k_e, s(k_d)\rangle$, and $\langle J, k_j, s(k_e)\rangle$, respectively. When node A query node J about is next vertex to k_i and node J replies with the corresponding certificate $\langle I, k_i, s(k_j)\rangle$, A knows that it has found a path to k_i and there is no need to query node I any further because A already has all the required certificates. The dashed line in figure 6.6 is the path from k_a to k_i and represents those certificates that A has collected.

One interesting property of the OCPD process is that it always will find the shortest certificate path between two nodes. This is because the outgoing path table maintained at each node always stores the shortest path information to reach other nodes. It makes the OCPD process very efficient. It enables us to find the needed certificate path by querying the minimum number of nodes and delivering the minimum number of certificates. And because the certificate path table is constantly updated, the OCPD mechanism will always be able to find a certificate path between two nodes provided there is one.

6.7 Discussions

6.7.1 Network partitions

Network partitions can happen frequently in MANET because the wireless link is easily broken. If two network nodes are in different partitions, then they can't communicate with each other. We consider different cases of network partitions and will discuss how they may affect the function of public-key authentication in mobile ad-hoc network.

For any two nodes within the same partition, the function to authenticate each other's public key using either MSA or MDA will not be affected because the exchange of certificate repositories only happens between the two nodes. If two nodes are in different partitions, then the proper operation of MSA or MDA will be affected. Because in this case they can't communicate with each other, they are not able to send their certificate repositories to each other to perform public-key authentication. A partial solution to this problem is to attach the node's local certificate repository to the information that needs to be authenticated. For example, if Alice wants to create an authenticated document and deliver it to other nodes, she can attach all certificates in her local certificate repository with the signed document. When Bob gets this document, Bob doesn't need to contact Alice to authenticate Alice public key. He can merge the set of certificates attached to the document and the set of certificates in his local certificate repository, and try to find a certificate path from himself to Alice. This equals to the case that *Alice* sends her local certificate repository to *Bob*. It enables the receiver to authenticate the signer's public key without contacting her. While this approach is useful for integrity checking, data authentication and non-repudiation purposes with the overhead of attaching certificates to the signed information, it doesn't solve all the problems. For confidentiality purposes, the receiver's public key is needed prior to performing the encryption function. For example, if

Alice wants to encrypt a document that is intended to be received by Bob, she will need to authenticate Bob's public key before she can do the encryption. Attaching the certificates in her local certificate repository to the document will help nothing in this case.

In this situation, it would be useful to use the OCPD process to authenticate the receiver's public key. Whenever a user needs to authenticate the other user's public key the user can use OCPD mechanism to discover a certificate path to the other user and authenticate its public key. The convenience of the OCPD process is, there is no need to contact the user whose public key needs to be authenticated. Thus even in the case that the receiver is not in the same partition or is offline, we are still able to authenticate its public key. This is another reason why OCPD is useful.

An extreme case can happen when some intermediate nodes are not online in the OCPD process. In OCPD, we need to query each node on the shortest path to the public key that needs to be authenticated. If one of these nodes is not available, then clearly we can't get the required certificate and find a certificate path to the public key we want to authenticate. It is possible alternative paths exist, but we don't have that knowledge because the certificate path tables always maintains the shortest path information. In this case, we still wish to be able to find a path if there exists one. The following mechanism can be used to serve this purpose. In the CPE process, neighbor nodes periodically exchange certificate path tables with each other. We propose to add the following rule to the process. If a node doesn't receive the outgoing certificate path table from an outgoing neighbor node in the certificate graph in ζ time, then the node will assume this outgoing neighbor node is not available and delete it from the next column in its outgoing path tables. ζ is a system parameter to define the length of time that a node should wait for another node's update before assuming it is not available. The node's outgoing path tables will then be updated and alternative paths will be found by the CPE process if there are some. By this mechanism, the certificate path tables built by the CPE process will reflect node's availability due to network partitions. The MSA and OCPD process will be able to get the required certificates based on this information.

6.7.2 Issuing and revoking certificates

Issuing certificates can happen very often in the MANET. Two existing nodes may issue new certificates to each other. A node newly joining the network may issue certificate to an existing node, and vice versa. Certificate revocation is to revoke an existing valid certificate and may also happen. Issuing and revoking certificates will change the certificate graph. Issuing a certificate corresponds to adding an edge to the certificate graph and revoking certificate corresponds to deleting an edge from it. When a new certificate is issued or an existing certificate is revoked, the issuer of the certificate will modify the corresponding entries in its certificate path tables. Issuing a certificate corresponds to adding a new entry in the outgoing certificate path table, and revoking one corresponds to deleting the correspond entry. The modification of the certificate path table will be populated to other nodes using the CPE process. After the "convergence time", all nodes in the network will learn this modification and have the most updated information.

6.7.3 Multiple certificate paths

The problem and solution presented in this paper are mainly targeted at how to enable users to authenticate each other's public key. This authentication is done via a single certificate chain. As we have shown in chapter 3 and 4, multiple certificate chains are always enhancements to the authentication of public keys. We considered this issue and did some modification to algorithm 7 so it may be used to collect certificates in multiple paths and construct multiple certificate chains between any two nodes. The modification is simple. In line 3 of algorithm 7, given the required number ϕ of certificate chains, we simply push k_x ϕ times into S. By doing this, the algorithm will do ϕ times of DFS searches in parallel. It basically means the depth of each DFS search will be synchronized. The result is ϕ DFS trees all starting from the public key of the node who is collecting the certificates. The edges in these multiple DFS trees represent those certificates that need to be collected.

We did some experiment to measure the performance of the modified algorithm of constructing multiple certificate chains. At this time, only performance for $\phi = 2$ is available. That is, in the merged certificate repository of any two nodes, there are two disjoint certificate chains between them. The performance results are shown in table 6.14.

In table 6.14, we can see the algorithm works fine for $\phi = 2$. The number of certificates that each node needs to collect are still very reasonable. For a small keyring with 27 keys and 160 certificates, only 8 certificates are required. For a large keyring with 588 keys and 5237 certificates, 20 certificates are sufficient for any two nodes to construct two disjoint certificate chains between them.

keyring	KR26-259	KR27-160	KR27-554	KR27-163	KR28-250
λ	12	16	8	12	8
keyring	KR30-177	KR32-202	KR588-5237	KR34-520	KR33-164
λ	12	12	20	12	12

Table 6.14: Minimal λ for any node to construct two disjoint certificate chains to any other node

6.8 Conclusion

In this work, we have presented the problems of the application of webs of trust in MANET. The distributed nature of MANET makes it particularly hard to provide reliable webs of trust computation. We discussed these challenges and the approaches that are suitable for it. We improved an existing approach and provide a much efficient solution. We also analyzed the optimal solution and were able to get a lower bound of the solution. Experiments showed that our solution was very close to the lower bound. We considered the case of network partitions and showed how our solution could deal with such situations. Both the analysis and the experiment results showed that our algorithm performs much better than the existing one.

Chapter 7

Conclusion and future work

7.1 Conclusion

Webs of trust system are very interesting distributed trust systems. In this system, users help each other and depend on each other to authenticate their public keys. They can construct the system by themselves without any outside assistance. The cost to maintain such a system can almost be ignored because it is quite evenly distributed to each individual user. While the distributed nature of webs of trust systems benefit it greatly, it also makes it vulnerable to malicious user attacks. The way that certificates can be freely issued makes it possible that unreliable users will certify incorrect public keys.

The works presented in this paper are used to solve this problem. By making use of them, we are able to construct a very robust webs of trust system that is resistant to malicious attacks. Our main contributions in this paper are as follows. First, we pointed out that malicious users can launch effective attacks by possessing multiple public keys or claiming multiple false identities. Our suggestion is to use multiple identity-disjoint certificate chains as a measurement in computing the trust. In addition, we addressed the case of certificate conflict the first time. We showed that the certificate conflict can actually be used to improve the robustness of webs of trust. Second, we showed that the current real-world webs of trust system PGP is very weakly protected and vulnerable to attacks. We presented a close-to-optimal heuristic to recommend users to issue additional certificates to make very robust the system. To make our method more practical, we also considered user's non-compliance and preference in these recommendations. Third, we were looking at how the certificate recommendations can be made more efficient. We presented a new way of random certificate recommendations, which can be used to construct a very robust webs of trust system, with only a moderate number of additional certificates, in the face of a large amount of malicious users. Using this method, we also presented an algorithm on how to detect malicious users. Experiment results showed that it is a very effective method. In the last, we considered the application of webs of trust system in the context of ad-hoc networks. We borrowed the idea of routing protocols and applied it in the webs of trust system. The result is a more efficient method that is used to authenticate public keys in the mobile ad-hoc network. For the method to be more robust, we also consider the issues of network partitions and multiple certificate chains. Our method is shown experimentally very effective and robust in the mobile ad-hoc networks.

7.2 Future work

7.2.1 Certificate conflicts

Certificate conflicts are very interesting. Their may be multiple reasons for certificate conflicts to happen. The current PGP webs of trust allows user to have multiple public keys. If a user create multiple public keys for herself, they may produce conflicts but they are all legitimate public keys. If a malicious user creates a public key and associate it with another user in a certificate, then it is going to be conflict with the legitimate public key for that user. In this case, they are not both legitimate public keys. The two cases are different but it is hard to differentiate them without contacting the corresponding user. There are two possible solutions to this problem. One is that we index each public key; like what we have suggested in chapter 4. By indexing each public key, all legitimate public keys which belong to a single user will not produce any conflicts. If a malicious user creates a public key and assign it with the same index number as a legitimate public key, then it is going to produce a conflict. Another possibility is to have the user cross sign all their public keys. In this way, though a user has multiple public keys, they will cross sign each other and when they do, we can assume that they are not conflicting with each other. The false public key created by the unreliable user will not be able to be cross signed by the user's legitimate public key. Using these methods, users can have multiple public keys without creating conflicts. When there are conflicts, we know that it must be created by the unreliable user's false public keys. Thus, users should be cautious of any public keys that are in conflict. We have presented several techniques in this work to detect unreliable users using certificate conflicts. It would be interesting to go further into this direction. These certificate conflicts can be used as evidence against unreliable users and they may be subject to penalty or be excluded from the webs of trust community if they are found to be malicious. It would also be interesting to see how certificate conflicts may be used in the real-world webs of trust system PGP, by applying the suspect set construct rules presented in this paper.

7.2.2 Certificate recommendations

We have presented two different certificate recommendation methods in this work. They have addressed a very serious problem of the current webs of trust system, i.e., it is very weakly protected against malicious attacks. They represent good directions in making more robust the webs of trust system. However, we believe that these recommendation methods can be tuned more and finely-grained to make better recommendations. For the recommendation method presented in chapter 4, it is possible to make better recommendations if we take users trust relationships into consideration. For example, if Alice believes what Bob believes for the authenticity of public keys, then there is no need to construct multiple certificate chains from Alice to all other users. Instead, as long as Bob can verify the authenticity of all public keys, Alice should too. In this way, there is no need for Alice to issue additional certificates at all. It would also be very interesting to see how the recommendation works by integrating them into the real world webs of trust system. For the random recommendation method, we can consider the possibility of introducing weight into the recommendations. If the certificate issued by different users can be weighted, then we the recommendation may be made more efficient and require fewer additional certificates.

7.2.3 Webs of trust in MANET and sensor networks

Webs of trust appear to be a suitable public-key infrastructure system for mobile ad-hoc networks because they are both distributed. We have shown a method that can help users to authenticate each other's public key in the MANET. The presented method can enable users to authenticate each other's public key via a single certificate chain. In terms of security, it would be much better if we can authenticate user's public keys using multiple certificate chains. We have presented some good results on constructing two disjoint certificate chains between users, it would be very interesting to develop a method that can construct more than two disjoint certificate chains.

It would also be very interesting to find out how the webs of trust system may apply in the sensor networks. It is usually believed that public-key infrastructures may not apply in sensor networks because sensors can't afford to expensive public-key operations. However, it can expected that more powerful sensor nodes will be introduced in the near future. And an efficient public-key algorithm elliptic curve[62] can be an ideal candidate for the sensor networks. It can be foreseen that public-key cryptography could be applied in sensor networks in the near future. Sensor networks are different from MANET since theyre mostly static and may contain far more nodes than MANET. They may be deployed on the field and may be easily captured. How to apply webs of trust in sensor networks could be a very interesting future work.

Bibliography

[1] Cacert.org. http://www.cacert.org.

[2] Verisign, inc. http://www.verisign.com.

[3] *Digital signature standard (DSS).* National Institute of Standards and Technology, Washington, 2000. URL: `http://csrc.nist.gov/publications/fips/`. Note: Federal Information Processing Standard 186-2.

[4] L.A. Adamic and B.A. Huberman. Zipf's law and the internet. *Glottometrics*, 3:143–150, 2002.

[5] R. Ahuja, T. Magnanti, and J. Orlin. *Network flows : theory, algorithms, and applications.* Prentice Hall, Englewood Cliffs, N.J., 1993.

[6] R. Albert, H. Jeong, , and A.-L. Barabsi. Diameter of the world wide web. *Nature*, 401:130–131, 1999.

[7] Motoi Aoki, Masato Saito, Hiroto Aida, and Hideyuki Tokuda. Anarch: A name resolution scheme for mobile ad hoc networks. In *AINA '03: Proceedings of the 17th International Conference on Advanced Information Networking and Applications*, page 723, Washington, DC, USA, 2003. IEEE Computer Society.

[8] Tuomas Aura. On the structure of delegation networks. In *Proc. 11th IEEE Computer Security Foundations Workshop*, pages 14–26, Rockport, MA, June 1998. IEEE Computer Society Press.

[9] A. Barabsi, R. Albert, H. Jeong, and G. Bianconi. Power-law distribution of the world wide web. *Science*, page 287, 2000.

[10] Andrs A. Benczr. Pushdown-reduce: an algorithm for connectivity augmentation and poset covering problems. *Discrete Applied Mathematics*, 129(2-3):233–262, 2003.

[11] Dimitri Bertsekas and Robert Gallager. *Data networks.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987.

[12] Thomas Beth, Malte Borcherding, and Birgit Klein. Valuation of trust in open networks. In *Proceeding of the 3rd European Symposium on Research in Computer Security (ESORICS 94)*, pages 3–18, 1994.

[13] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173, Oakland CA USA, 6-8 May 1996.

[14] Mike Burmester and Yvo G. Desmedt. Is hierarchical public-key certification the next target for hackers? *Commun. ACM*, 47(8):68–74, 2004.

[15] S. Capkun, L. Buttyan, and J. P. Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(1):17, March 2003.

[16] Srdjan Capkun, Levente Buttyn, and Jean-Pierre Hubaux. Small worlds in security systems: an analysis of the pgp certificate graph. In *Proceedings of the 2002 workshop on New security paradigms*, pages 28–35. ACM Press, 2002.

[17] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill, second edition, 2001.

[18] Dave Darnell. Pgp or pki? the future of internet security. *EDI Forum: The Journal of Electronic Commerce*, 12(1):59–62, 1999.

[19] Yvo G. Desmedt and Yair Frankel. Threshold cryptosystems. In *Proceedings on Advances in cryptology*, pages 307–315. Springer-Verlag New York, Inc., 1989.

[20] John R. Douceur. The sybil attack. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, MIT Faculty Club, Cambridge, MA, USA, March 2002.

[21] Wenliang Du, Jing Deng, Yunghsiang S. Han, and Pramod K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *Proceedings of the 10th ACM conference on Computer and communication security*, pages 42–51. ACM Press, 2003.

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

[23] Laurent Eschenauer and Virgil D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 41–47. ACM Press, 2002.

[24] A. Frank. Connectivity augmentation problems in network design. *Mathematical Programming: State of the Art 1994*, pages 34–63, 1994.

[25] A. Frank and T. Jordan. Minimal edge-coverings of pairs of sets. *Journal of Combinatorial Theory, Series B*, 65(1):73–110, 1995.

[26] Andras Frank and Tibor Jordan. Directed vertex-connectivity augmentation. *Mathematical Programming*, 84(3):537–553, 1999.

[27] J. J. Garcia-Luna-Aceves. A minimum-hop routing algorithm bases on distributed information. *Comput. Netw. ISDN Syst.*, 16(5):367–382, 1989.

[28] J. J. Garcia-Luna-Aceves. A unified approach to loop-free routing using distance vectors or link states. In *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*, pages 212–223, New York, NY, USA, 1989. ACM Press.

[29] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W H Freeman & Co., 1979.

[30] Frank Harary. *Graph Theory.* Addison-Wesley, Reading, Mass., 1969.

[31] C. Hendrick. Routing information protocol. RFC 1058, IETF, June 1988.

[32] He Huang and Shyhtsun Felix Wu. An approach to certificate path discovery in mobile ad hoc networks. In *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 41–52. ACM Press, 2003.

[33] Jean-Pierre Hubaux, Levente Buttyan, and Srdan Capkun. The quest for security in mobile ad hoc networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 146–155. ACM Press, 2001.

[34] I. Itai, Y. Perl, and Y. Shiloach. The complexity of finding maximum disjoint paths with length constraints. *Networks*, 12:277–286, 1982.

[35] Qinglin Jiang, Douglas S. Reeves, and Peng Ning. Certificate recommendations to improve the robustness of webs of trust. In *International Security Conference 2004*, volume 3225 of *Lecture Notes in Computer Science*, pages 292–303. Springer, 2004.

[36] Qinglin Jiang, Douglas S. Reeves, and Peng Ning. Improving robustness of PGP by conflict detection. In *The Cryptographers Track at the RSA Conference 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 194–207. Springer, 2004.

[37] D.S. Johnson. Worst case behavior of graph coloring algorithms. In *Proc. Fifth Southeastern Conf. Combinatorics, Graph Theory, and Computing*, pages 513–527, 1974.

[38] Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh V. Vazirani. On syntactic versus computational views of approximability. In *IEEE Symposium on Foundations of Computer Science*, pages 819–830, 1994.

[39] Jon Kleinberg. The small-world phenomenon: an algorithm perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170. ACM Press, 2000.

[40] James F. Kurose and Keith Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[41] Butler Lampson, Martin Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: theory and practice. *ACM Transactions on Computer Systems (TOCS)*, 10(4):265–310, 1992.

[42] R. Levien and A. Aiken. Attack-resistant trust metrics for public key certification. In *Proceedings of the Seventh USENIX Security Symposium*, 1998.

[43] Donggang Liu and Peng Ning. Establishing pairwise keys in distributed sensor networks. In *Proceedings of the 10th ACM conference on Computer and communication security*, pages 52–61. ACM Press, 2003.

[44] Haiyun Luo, Petros Zefros, Jiejun Kong, Songwu Lu, and Lixia Zhang. Self-securing ad hoc wireless networks. In *Seventh IEEE Symposium on Computers and Communications (ISCC '02)*, 2002.

[45] Ueli Maurer. Modelling a public-key infrastructure. In *Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS 96)*, pages 324–350, 1996.

[46] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: Universal topology generation from a user's perspective. Technical Report BU-CS-TR-2001-003, Boston University, 2001.

[47] S. Mendes and C. Huitema. A new approach to the X.509 framework: Allowing a global authentication infrastructure without a global trust model. In *Proceedings of the Symposium on Network and Distributed System Security, 1995*, pages 172–189, San Diego, CA , USA, Feb 1995.

[48] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of applied cryptography.* CRC Press, Boca Raton, Fla., 1997.

[49] M.Grotschel, L.Lovasz, and A.Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169 –197, 1981.

[50] Hiroshi Nagamochi and Toshihide Ibaraki. Graph connectivity and its augmentation: applications of ma orderings. *Discrete Appl. Math.*, 123(1-3):447–472, 2002.

[51] Charles E. Perkins. Ad hoc networking: an introduction. pages 1–28, 2001.

[52] M. Reiter and S. Stubblebine. Toward acceptable metrics of authentication. In *IEEE Symposium on Security and Privacy*, pages 10–20, 1997.

[53] M. Reiter and S. Stubblebine. Resilient authentication using path independence. *IEEE Transactions on Computers*, 47(12), December 1998.

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

[55] Michael K. Reiter and Stuart G. Stubblebine. Path independence for authentication in large-scale systems. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 57–66. ACM Press, 1997.

[56] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[57] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[58] M. Drew Streib. Keyanalyze - analysis of a large OpenPGP ring. http://www.dtype.org/keyanalyze/.

[59] Anas Tarah and Christian Huitema. Associating metrics to certification paths. In *Computer Security - ESORICS 92, Second European Symposium on Research in Computer Security, Toulouse, France, November 23-25, 1992, Proceedings*, volume 648 of *Lecture Notes in Computer Science*, pages 175–189. Springer Verlag, 1992.

[60] Int'l Telecommunications Union/ITU Telegraph & Tel. ITU-T recommendation X.509: The directory: Public-key and attribute certificate frameworks, Mar 2000.

[61] D. Watts and S. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440, 1998.

[62] ANSI X9.62. "The elliptic curve digital signature algorithm (ECDSA)". American Bankers Association, 1999.

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

[64] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. Leap: efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM conference on Computer and communication security*, pages 62–72. ACM Press, 2003.

[65] Philip Zimmermann. *The official PGP user's guide.* MIT Press, Cambridge, Mass., 1995.