Probabilistic public-key confidence valuation model for a Peer to Peer PKI

Tomasz Cholewiński

LYNGBY 2004
MSc THESIS
NR. 47/2004

IMM

Trykt af IMM, DTU

Preface

This Master's thesis is the result of work carried out from 02.02.2004 to 12.07.2004 at the department of Informatics and Mathematical Modeling of the Technical University of Denmark (DTU).

Intended audience

This thesis is not only a theoretical study but it also aims to provide a set of library routines that could serve as a basis for developing a practical Peer 2 Peer PKI implementation.

Readers are assumed to have a basic knowledge of Java and Object Oriented Programming (OOP) required to understand the code excerpts and algorithms presented in this thesis.

It is also helpful if the reader is familiar with the concepts of Peer to Peer, Public Key Infrastructure, Authenticity and Trust. However the thesis does give a cursory introduction to each of those topics along the way.

Acknowledgments

The work has been done under the supervision and guidance of Associate Professor Christian Damsgaard Jensen, to whom I wish to extend thanks for feedback, comments and support.

Abstract

Peer to Peer systems are becoming widespread throughout the Internet and pervasive computing systems. Existing PKI infrastructures - both hierarchical and non-hierarchical cannot be directly ported to P2P environments. This is because existing PKIs rely heavily on the presence of CAs, which act as a "trusted third party" in the system. The problem of feasibility of implementation of various functions performed by PKI systems in such an environment is analyzed.

The goal of this thesis is to explore the possibility of implementing a Peer to Peer PKI system based on the idea behind the PGP Web of Trust and a probabilistic algorithm to evaluate the confidence in a public-key from a CA, which is described in the paper "Modeling a PKI Infrastructure" by Ueli Maurer [19]. The public-key valuation model makes assumptions about trust in the "trusted third party" explicit, which allows the system to use key-servers that are not completely trusted. This is particularly helpful in a wireless P2P environment targeted by this work.

The feasibility of implementation of the probabilistic confidence parameter valuation model is evaluated using a software prototype.

The conclusions drawn during the design and implementation phases of the prototype serve as a basis of an overall feasibility evaluation. A problem is identified involving the complexity of calculations of higher level trust paths. Further research paths are outlined, including sensitivity analysis for finding certification paths which contribute most to the end-value of the confidence parameter.

Keywords

Peer to Peer, P2P, Public Key Infrastructure, PKI, PGP Web of Trust, Trust, Authenticity, Java

Contents

Chapter 1

Introduction

1.1 Peer to Peer

The Internet and all of today's networks are changing. There has been a tremendous increase in both the required and supplied bandwidth - the networks have evolved from fixed line kilobit speed dialup connections to wired multigigabit fibre and multimegabit wireless connections. In 1962, the first commercially available modem boasted a transfer speed of 300 bits per second full duplex. In 2002 Fujitsu alone delivered a commercially available DWDM system - the FLASHWAVE 7700 capable of delivering 1.76 Terabits per second. Achievable speeds in Internet 2 [1] are heading into the Gbps range [2] - which usually means that transfer rates are limited by the speed of hard drives at the client machines.

This often means that the usual client-server concept is not applicable any more. An example of this is the so-called slashdot effect. The *slashdot.org* website is a technology-oriented weblog which delivers news and insight on current issues in the world of hi-tech. However due to the number of readers visiting the site (well over 700000) external websites included as part of the news entries often fail due to excessive load within minutes of the article being posted. This is called the *./ effect* (pronounced slashdot effect) and refers to a situation where the available server bandwidth or other resources (cpu time, available ports) are quickly consumed by large numbers of well connected (high bandwidth) clients. This leads to the slashdotted server appearing offline or unresponsive.

One example of an attempt to deal with the scalability issue is the evolution of file sharing networks - the usual approach calls for extensive distributed server architectures with dedicated high speed connections and clusters present where the query density is likely to be the largest. However such designs are costly and have to be hardened against hardware and software failures. The recent inexpensive approach calls for numerous simple clients to act as servers as well as clients thus participating in the load carrying capacity of the network. It requires no prior infrastructure, and the system scales proportionally to the number of users. Each new user contributes a part of his/her cpu time and network connection to enhance the capabilities of the entire system for the benefit of other users.

This situation, where a client can also play the role of the server simultaneously is at the very core of the definition of Peer to Peer. Although this thesis will mainly focus on P2P networks linking wireless devices in an ad-hoc way, it should be clear that P2P is a much broader concept - one that encompasses all kinds of wired/wireless and hybrid networks and a variety of protocols. Peer to Peer is further described in chapter 3 on page 23.

Throughout the following sections, the peers will be referred to as nodes, the communication between them will be assumed as wireless. The reason behind this is that ad-hoc wireless P2P networks are on the extreme end of dynamic networks, and the concepts derived here should be easily applicable to wired P2P networks.

1.2 Wireless

The concept of wireless communication is by no means new, but recent years have shown a huge improvement in wireless communication technology. The available networks range from cellular systems such as the GSM 900/1800 [5] and PCS 800/1900 [4] - digital circuit switched mobile telephony networks, GPRS [6], EDGE [7] - packet switched cellular data and UMTS [8], CDMA2000 [9] - packet switched cellular data/telephony networks. In the world of short range computer networks there are the radio frequency WLAN [10], Bluetooth [11], HomeRF (WLAN alternative, working group disbanded in 2003) [12] and infrared IrDA [13] networks. There are also very short range specialized wireless radio frequency modes of communication - RFID [15] and NFC [14]. In this variety of communication technologies there is great potential for making our lives easier. Cellular telephony is already taken for granted - most industrialized countries have more mobile phones than fixed line.

1.2.1 Wireless Infrastructure

The typical wireless infrastructure consists of mobile terminals - user equipment, and Base Station Transmitters - transmitter masts, that constitute a single or multiple *cells*. Telephone networks interconnect mobile devices with no infrastructure existing between the mobile phone and the BTS transmitter. This allows a high degree of flexibility, as deployment cost in sparsely populated areas is very low compared to wired infrastructure solutions. First generation networks offered only analogue connection setup and transmission. Second (GSM) and third (UMTS) generation networks offer full digital voice and data transmission functions. The only drawback is that user terminals (mobile phones) cannot directly communicate to each other even if they are in direct range. The connection setup phase always sets up the circuit through a BTS and call and billing center. Use of short-range ad-hoc transmission technologies is required to interconnect mobile devices without using any infrastructure (wired or wireless).

1.2.2 Ad-Hoc

Ad-hoc networks are a relatively new concept which appeared with the introduction of radio frequency and infrared wireless communication modes for mobile devices. An ad-hoc network can be defined as a network that doesn't require existing infrastructure in order to work. This opens up a host of possibilities such as ad-hoc networking in emergency situations, disaster recovery, medical facilities, battle zones and ad-hoc sensor dust deployment.

An ad-hoc network is merely a collection (be it physical or logical) of mobile devices, when two users wishing to exchange information need not be in range of themselves, just other devices in the network. As long as there are other nodes in the ad-hoc construct that can relay information further on, the notion of ad-hoc networking is preserved. Technology-wise - WLAN, Bluetooth, HomeRF, IrDA are a means of connecting mobile devices that are in range of the built-in transceiver. With an added layer of software routing, true ad-hoc can be achieved

No infrastructure, no connection to an extranet, and non homogeneous participating devices mean that such an environment is unique in terms of many of the paradigms we take for granted on the Internet. One of them is the issue of creating and maintaining an efficient PKI.

Ad-hoc networks are further discussed in section 3.2 on page 26.

1.3 Security in Mobile Ad hoc networks

All networks call for some kind of security which allows users to communicate without fear of being misguided or attacked. A PKI is usually the prerequisite to authentication and authorization of users in networks where company secrets and money are involved. PKI itself deals with the issuing and verifying certificates. A *certificate* is a binding between an entity and a public key. It is a way of ensuring that the public key corresponds to the individual or entity with which we wish to communicate. How the PKIs achieve the objective of positive identification and verification of individuals or entities is implementation specific and is discussed in detail in chapter 2 on page 15.

1.4 Peer to Peer Public Key Infrastructure

This thesis establishes a theoretical foundation covering the evolution of the Public Key Infrastructure, the Peer to Peer paradigm and Ad-Hoc networks. This foundation is then used to introduce a novel approach to designing an ad-hoc PKI - probabilistic confidence parameter valuation [19]. A software prototype is then designed and developed in order to prepare a feasibility evaluation of the concept. This thesis focuses on the PKI in the context of ad-hoc networks. Such an implementation is particularly challenging, as present day hierarchical PKIs require fixed, dependable infrastructure to be in place in order to function. Additionally a reliable connection to that infrastructure is required for each participant. Ad hoc environments offer no fixed, predictable infrastructure that would allow such an implementation. Additionally in the fixed client server world clients can usually assume that the server is trustworthy, while in the mobile ad-hoc world, each node is potentially hostile. A new kind of approach for realizing a PKI is chosen and examined. Various approaches including the chosen probabilistic trust algorithm by Ueli Maurer [19], section 2.3.3 are presented in chapters 3 and 4. The chosen algorithm is used as a basis for the design of a Java library in chapter 5. The following chapter - chapter 6 on page 45 discusses the Java implementation of the Ad hoc PKI algorithm. Finally, the design and implementation are evaluated in chapter 7 on page 53 and a summary is presented in chapter 8 on page 55. The source code is presented in Appendix A.

Chapter 2

Public Key Infrastructure

2.1 Introduction

Traditional symmetric cryptography calls for the sender and receiver of a message to share a cryptographic key. This prior requirement often creates a problem if the two parties do not have a secure way of establishing the cryptographic key. This is known as the key distribution problem. Should an attacker intercept the initial communication and obtain the key, then he can easily read the messages exchanged by the two parties. Solutions to this problem involve key agreement algorithms - such as the Diffie-Hellman exchange, or key distribution through secure out-of-band methods. Keys could be for example exchanged using courier, in person or through secure delivery services. However this all means that cryptography is well out of reach of ordinary people. The necessary breakthrough is to devise a method of transmitting a cryptographic key over an insecure channel.
The concept of the PKI or *Public Key Infrastructure* dates back to a revolutionary 1976 paper by Whitfield Diffie and Martin E. Hellman [17]. There they introduce the foundations of public key cryptography, which essentially allows people to go around the secret key distribution problem. A cryptographic key would essentially be split into a public and private key. The public key could be made readily available to anyone who asked, and freely transmitted over insecure channels. The private key would have to be held more secret than a ATM card PIN - it would never have to leave the user's system. The first published public key algorithm is RSA in a 1978 paper by Rivest, Shamir and Adleman [25]. It is based on a simple mathematical transformation:

- Key Generation
  Generate two large primes p and q
  $n = p \cdot q$
  $m = (p-1)(q-1)$
  Choose e, coprime to m
  Find d, such that $(d \cdot e) mod m = 1$
  $[e, n]$ - public key
  $[d, n]$ - private key

- Encryption $C = P^e mod n$

- Decryption $P = P^d mod n$

In order to send an encrypted message to a user Bob, Alice would have to obtain Bob's public key, and encrypt the message using that key. Once she did that, the only person that can decipher the message is Bob, with his corresponding private key. As a consequence Diffie and Hellman suggested the notion of a public key distribution system called the *Public File*. This system would allow the lookup of a public key corresponding to a name in the system - a way for Alice to obtain Bob's public key. This kind of repository roughly corresponds to today's *Trusted Repositories* - an entity that signs all of its communication with the user.
There are essentially a number of problems associated with the Public File. The repository has to be secured against all possible intrusions, as an attacker could try and modify the existing public keys, so that they corresponded to his own private key. Additionally the repository would have to be prepared to take a large load of user queries - each attempt at encrypted communication would have to result in a query to the online repository. The solution to these problems is addressed as an extension of the concept of the Public File in a 1978 B.Sc. paper by Loren M. Kohnfelder [18]. Kohnfelder introduces the concept of an identity certificate, which is essentially a signed message from the Public File containing Bob's public key and a plaintext field containing Bob's name. This serves as a means to take the brunt of the security requirements from the Public File and shift it to the user. As each certificate is essentially self-contained, and it only requires access to the public key of the repository for verification purposes, the number of queries to the repository itself is reduced dramatically. Additionally the security requirements on the repository are substantially relaxed, only the private key used to sign the certificates needs to be guarded, so that it is not used to issue bogus certificates.
This chapter will introduce the basic concepts behind PKI using X.509 as the prime example, and then moving on to cover the more current developments in the world of Public Key Infrastructures.

2.2 X.509

The concept of a repository containing user certificates is the very foundation of the design of a Public Key Infrastructure. The first practical attempt at producing a worldwide public key infrastructure is part of the efforts to provide a secure access mode to X.500 directories. X.500 in essence a hierarchical database that would be administered by dominant telecommunications companies around the world.
At the core of the X.500 concept is the DN, or *Distinguished Name*, which is composed of RDN's, or *Relative Distinguished Names*, by traversing the hierarchy of the directory. An example is shown in figure 2.1.

The figure shows a sample X.500 tree, where at the top is the root node, then country level nodes (`C=`), organization level node (`O=`), organization unit node(`OU=`). Additional node types are of course possible, as X.500 was designed as a single global directory solution.
X.509 certificates essentially contained the issuer DN, the subject DN, a validity period and the public key itself. The problem with these kinds of certificates is that their rigid structure does not allow insertion of other useful information into the certificate - e.g. the subject's address, organizational structure. Additionally the DN is essentially meaningless outside the X.500 hierarchy. Given that X.500 directories were never widely deployed, this has led to the DN fields being misused in an attempt to provide unique names to each of the certificates. Another problem that the X.509 certificates face is the unclearly defined CRL's, or *Certificate Revocation Lists*. A more detailed view on the problems a PKI is likely to face is given below.

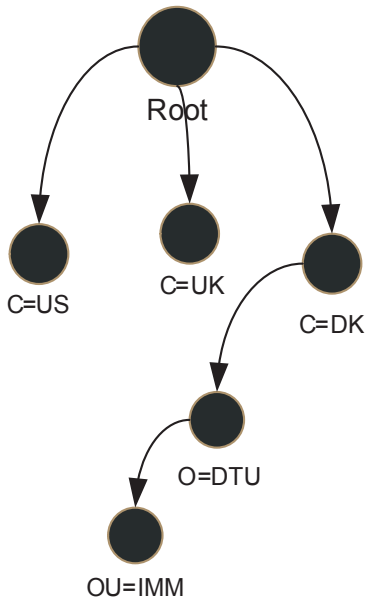

Figure 2.1: Sample X.500 tree

2.2.1 Certificate acquisition problem

As mentioned before, X.500 is not really deployed worldwide. This causes a number of problems with X.509 certificates, one of which is obtaining a certificate in the first place. The original model simply called for retrieving the certificate from a repository, however no centralized certificate repositories exist. Instead many solutions simply bundled the necessary certificates for issuers - e.g. a S/MIME signature includes the certificates that are needed for verification and a Secure Socket Layer [16] connection simply exchanges the necessary certificates during the initialization. Pretty Good Privacy [24] generally requires the user to obtain the certificate by some out of band means - e.g. by going to the recipient's website and downloading the certificate or by mailing the user asking for the certificate in advance.

However such offline and varied distribution methods make the hard problem of certificate revocation even harder.

2.2.2 Uniqueness of Distinguished Names problem

The integral part of any certificate is the name of the user that the public key belongs to. However, if the certificates are to be accepted worldwide, then the name of the user has to be unique. The best approach to this is to include a context along with the name. For example the name *John Smith* is essentially meaningless in the worldwide context, the person may be a resident or a former resident of the United States, or any other English speaking country. However if the name John Smith is paired with a unique identifier - such as an email, or social security number (along with the country that the social security number belongs to) the name becomes unique. The only other requirement is that a third party wanting to acquire a certificate for John Smith needs to be able to construct a *Distinguished Name* to single out the certificate that belongs to the particular John Smith that they want to talk to.

2.2.3 Certificate Revocation problem

However secure a system is designed to be, there are always weaknesses or mistakes made which lead to private keys being compromised. If such a situation occurs, it calls for declaring the corresponding certificate invalid - as attackers can easily take advantage of the leaked private key, and assume the identity of the legitimate user for fraudulent activities. The solution that X.509 uses is of CRL's, or *Certificate Revocation Lists*. These are simply lists containing the identifiers of certificates which are no longer considered valid. It is the responsibility of the application using X.509 certificates to retrieve a current CRL, before processing a certificate. As this solution is modeled after check and credit card black lists, it is fraught with problems in the digital age. For one is the question of availability - if each of millions of desktop users were to retrieve a CRL, the required distribution framework would have to rival many of the commercial giants such as Akamai [30] or Google [31]. There is also the question of time frame between new CRL's being issued - in the electronic age, it is theoretically possible that a leaked private key is used within a few seconds of the compromise taking place. However, most likely a CRL issuing time of 1 minute should stop most possible fraud (there is also the question of detecting that a certificate has been compromised). Even so, the distribution framework for CRL's would consume immense resources in terms of bandwidth and processing power - customers would have to pay for using such facilities. More realistically, CRL's are issued once a month or less often.

2.3 Present day Public Key Infrastructure

To date there exists no global PKI infrastructure. As X.500 has never been widely deployed, neither was X.509. Specialized PKIs have appeared, the most notable example being Netscape's website SSL [16] certificates based on X.509v3. These use the PKCS set of standards for the format of certificates, keys and digitally signed and encrypted objects and any of the available repository solutions - LDAP for example. Certificates are obtained by contacting a company that provides and maintains their own PKI infrastructure servers - examples are Verisign, Thawte Consulting, RSA Security Inc, etc. All mainstream SSL capable browsers come bundled with a complete list of global root authorities, maintained by the aforementioned companies, that can be used to verify a server certificate.

PEM is the solution that has adopted the X.509 approach for email. The original X.509 certificates are meant to be used to allow the user access to the corresponding X.500 subtree. PEM changes the focus of X.509 certificates for the purpose of identifying the user as the sender of a message.

2.3.1 PGP's Web of Trust

The main idea behind Philip Zimmerman's Web of Trust [3] is to become independent of rigid certification hierarchies. Figure 2.2 shows a side-by-side comparison between the certification hierarchy based on the X.509 concept and the Web of Trust concept. In the certification hierarchy, the certification server closest to the user issues the actual certificate, but the server certificate is signed and issued by a higher level certification server. Only the root server certificate is self-signed. In PGP certification, clients generate their own self-signed certificates, and certificate authenticity is verified by checking other user's signatures on a certificate.
PGP is introduced when there still was not a single global CA root that certificates could be

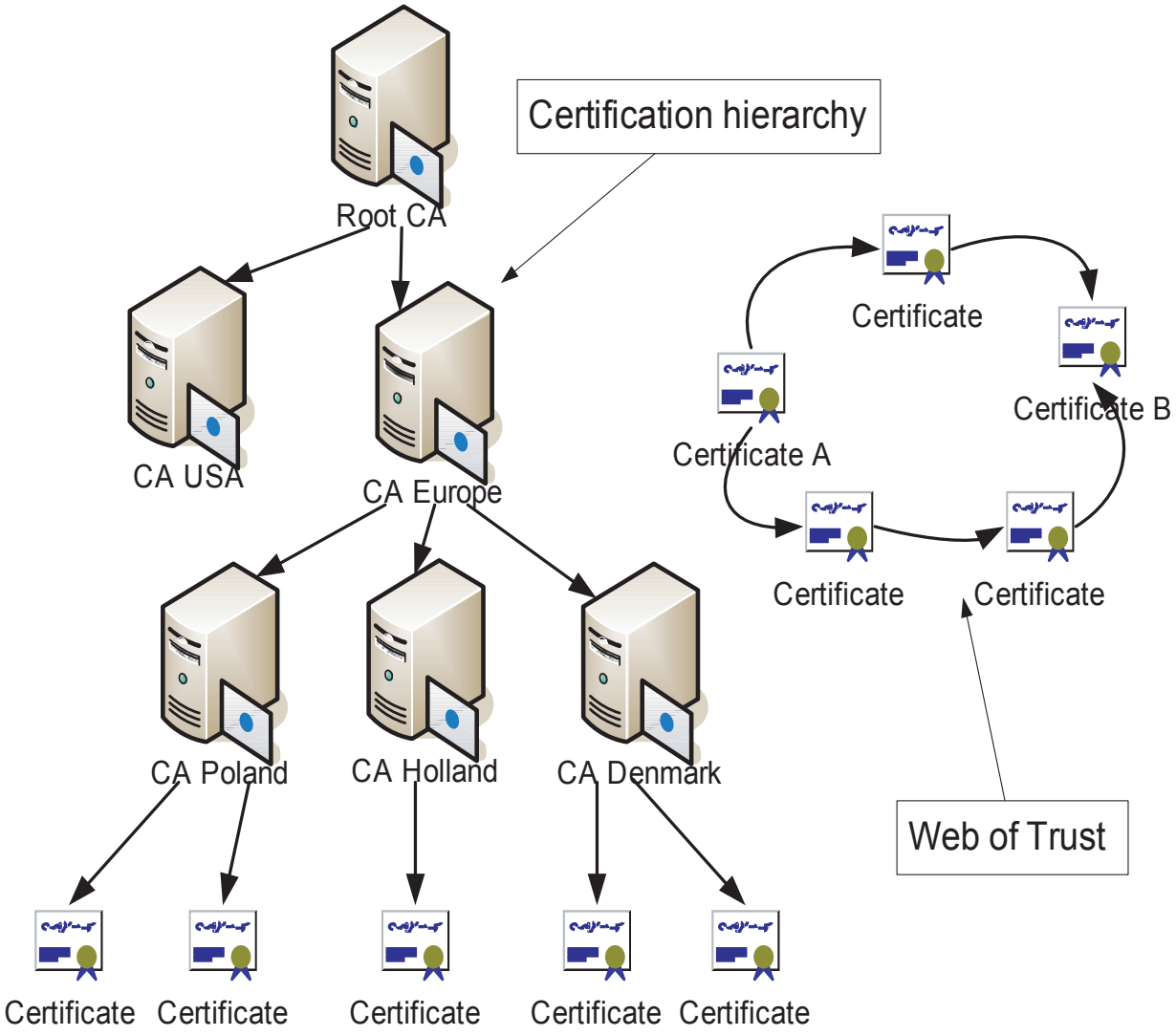

Figure 2.2: Certification hierarchy and Web of Trust

reliably signed by, and the chances for a CA like that appearing were rather small. An X.509 certificate is signed with one or more CA's private keys and to verify the certificate it is enough to retrieve the public keys of the individual CA's and check their signatures on Bob's certificate. In the Web of Trust, one first has to obtain the certificate from a certificate store, for example by sending an email with the subject

```
GET prz@mit.edu
```

to the email address

```
pgp-public-keys@keys.pgp.net
```

the response, given that such an ID exists in the database, should be an ASCII encoded version of the certificate of Philip Zimmerman [23]. However, there exists no guarantee that the key was actually created by Philip. As the email is never verified at any stage (nor is there any reliable method of verification), any individual could have created a public key with the name Philip Zimmermann, email prz@mit.edu and submitted it to the PGP keystore. Therefore before using the certificate to encrypt a message to the creator of PGP, one should verify the authenticity of

the certificate. This could for example be done by out of band means such as a phone call or a visit to Philip Zimmermann to obtain and check the public key properties. These properties include a human readable hash of the key (called a fingerprint), key length and key id. However this could prove to be difficult if one lives on a different continent than the person I wish to email. Although the website of Philip Zimmerman does list a fingerprint of the certificate, HTTP is in itself not a secure protocol, and a man-in-the middle attack is possible if unlikely. Fortunately PGP offers another method of verifying the authenticity of a certificate. Provided I have a trusted certificate of a person say Bob, that has verified and and put a signature on Phil Zimmerman's certificate, I can assign a degree of trust to the certificate in question. This can extend to a longer chain of "acquaintances" effectively forming a certification chain. This is the basis of the Web of Trust. Multiple certification paths may exist, and it is possible to define how many certification paths there need to exist to make a certificate trusted. This is done in belief that if a percentage of the parties we are dealing with is corrupt, then there may exist a certification path vouching for the authenticity of an otherwise fake certificate. However it is a viable assumption that not all parties will be corrupt, and that genuine certification paths will exist.
One of the key differences between X.509 and PGP is that PGP is primarily meant for the task of digitally signing and encrypting email messages. Although the recent releases by the PGP Corporation [24] try to adapt PGP to tasks such as file and partition encryption, the standard remains very focused. The next section discusses one of the alternatives to PGP and X.509 that tries to combine simplicity of application and flexibility - the ability to adapt to a variety of environments, not just digital directories or electronic mail.

2.3.2 Simple Distributed Security Infrastructure and Simple Distributed Key Infrastructure

Simple Distributed Security Infrastructure or *SDSI* is a new PKI concept that avoids using global certificate hierarchies and globally unique names (in the X.500 sense). It is proposed in 1996 by Ronald Rivest and Butler Lampson [21] as an alternative to the inflexible and overly complicated X.509 hierarchy and as a more general solution than the Web of Trust.
Similarly to PGP's Web of Trust, there is no predefined hierarchy and the system relies on keys alone to provide the functionality. The central part of SDSI is the notion of a principal which is essentially a designated public key used to verify the statements of the principal. All principals are made equal in terms of functionality - they act as peers and can provide and request signed statements from other principals. The issue of providing globally unique names is solved by local name spaces and designated *distinguished root* principals. Functionality similar to PGP's email-based unique names is obtained by using the DNS distinguished root principal, so that an email address:

`rivest@mit.edu`

becomes:

```
( ref: DNS!! edu mit rivest )
```

In the above, `DNS` is the name of the principal and the `!!` at the end indicates that `DNS` is a distinguished root principal. Such principals are always referred to by the same name from the other principals. This is as opposed to normal principals, which can be referred to by different (possibly conflicting) local names from different principal namespaces. Distinguished root principals exist to provide for global uniquely resolvable names.
Simple Public Key Infrastructure (SPKI) by C. Ellison et al. [22] builds on the notion of SDSI

and allows the name spaces and authorization based on principals. SDSI only had *identity* and *membership* certificates, while SPKI uses *naming*, *authorization* certificates, *certificate revocation* and *revalidation lists* (CRLs). A basic SPKI authorization certificate is a mapping of the form:

```
authorization -> key
```

where the keyholder's name is provided as an optional naming certificate mapping of the form:

```
authorization -> key -> name
```

The separation of the mappings allow for more security, as only the authorization mapping needs to be kept absolutely secure. Additionally SPKI defines ACL's more arbitrarily than SDSI. In the former, each entry in the ACL need not be a subject name, it may be the subject key or be implementation dependent.

2.3.3 Probabilistic Trust

Another approach to developing a PKI without a fixed infrastructure is the one proposed by Ueli Maurer [19] in 1996. The paper does not define a full working PKI infrastructure, instead it focuses on the reasoning that can be used to verify the authenticity of a public key. The concept builds on the Web of Trust in the sense that certification paths are considered in verifying the authenticity of public keys. However Ueli Maurer states that authenticity and trust are rarely absolutes - it is not always possible to say that something is fully authentic, or definitely fake. In real life we base our decisions on partial beliefs - e.g. we believe that someone is who he claims he is, but stay vigilant. Therefore trust and authenticity ranges from completely untrusted and unauthentic to fully trusted and authentic. It may be useful to assign a number between 0 and 1 to such statements - Ueli Maurers paper describes an algorithm that gives meaning to these "weights" as a probability of a statement being true in a well defined random experiment.
PGP's Web of Trust a user can define the minimum number of existing certification paths to cause a particular public key to be trusted. On the other hand, the certification paths may be interdependent, cyclic and contain relationships that cannot be captured by the Web of Trust (the users in seemingly disjoint certification paths may belong to the same organization and if one is corrupt, the other users in the organization are more than likely to be corrupt as well). Ueli Maurer's paper defines a method that is general enough to capture such interrelationships. Additionally through the notion of trust, there need not exist full certification paths from Alice to Bob. Alice may trust another entity to "vouch" for another entity's certificate, thereby bridging any gaps in the certification path.
The Peer to Peer PKI implementation in this paper uses Ueli Maurer's algorithm as part of the solution. The rationale for this decision is given in section 4 and the detailed description of the algorithm in section 5.

2.4 Summary

This section introduces the basic concepts of asymmetric cryptography and Public Key Infrastructure as the solution to the key distribution problem. This approach is used in the X.500 standard in the form of th X.509 certificate standard. Problems faced by the PKI are presented in section 2.2:

- Certificate acquisition problem

- Uniqueness of Distinguished Names problem
- Certificate Revocation Problem

Although X.500 and X.509 were never widely deployed in their original form, many derived systems are created and used. X.500 was designed to be versatile, however in present day networks, it seems specialized systems are much more popular. Netscape's SSL certificates use X.509v3 certificates hierarchy for certifying websites. PGP's Web of Trust is a working solution for encrypting and signing e-mails and it doesn't require a certification hierarchy to work. SPKI is a specialized system for authorization, and similarly to PGP it doesn't require fixed certification hierarchies. Finally, probabilistic trust is introduced as a new concept for determining the authenticity of a public key in an infrastructure-less environment. The next chapter will focus on such environments - peer to peer systems, and chapter 4 discusses the concept of probabilistic trust in detail.

Chapter 3

Peer to Peer Systems

Peer to Peer is a relatively new concept that serves as an alternative to the traditional client-server model. In this model, each node takes on a role of client and server simultaneously. Each node is capable of performing a transaction acting as either server or client. Nodes in a Peer to Peer network can be very diverse in terms of capabilities, processing power, bandwidth available, etc. Each of them needs to be able to support at least a minimum set of functions of a particular P2P protocol.

Figure 3.1 compares the Peer to Peer and Client-server concepts. In Peer to Peer, there is no visible hierarchy - each device acts as a peer to other devices. Client-server, on the other hand has a clear distinction of roles played by the devices.

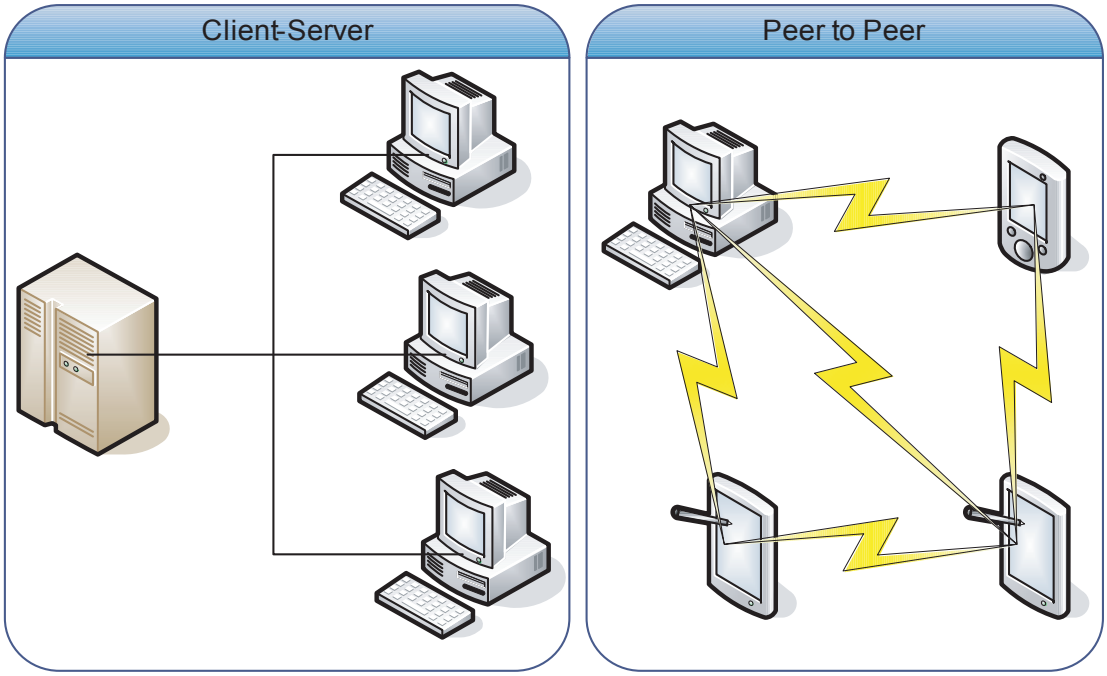

Figure 3.1: Client-Server and Peer to Peer

Currently the most widely known example of P2P applications are the filesharing networks on the Internet. What started as a client-server architecture with elements of P2P communication

- like Napster, has evolved to fully distributed Peer to Peer arrangements like Gnutella. Such networks have the attributes desirable to the application - they scale easily with the number of users increasing, the capabilities of the network increases as well, limited only by the efficiency of the protocol governing the transactions. Additionally P2P guarantees survivability - as file-sharing networks are often targeted by recording and film industry associations, due to copyright abuse, today's networks cannot be shut down, unless each and every node is disconnected from the network.

3.1 History

The Internet is originally intended to be a Peer to Peer network. As its original purpose is to connect academic centers as equals, each host would act as both a client and a server for ftp, email, etc. Although the protocols themselves are based on a client - server model, the data exchange is kept symmetric, hence the similarity to the P2P model. Later as the Internet grows larger and becomes more commercial, the client - server model becomes dominant. Most computers aren't expected to run ftp or http services, yet almost all of them use this services from servers located throughout the Internet. The inexpensive ADSL and Cable modem ISP connections further enhanced the notion that users download more than they upload - the typical upload speed of an ADSL model is 64/128 kbit/s, while the download speed can be of the order of 512/768 kbit/s. In essence users with their desktop computers were considered clients on the Internet. The Peer to Peer model is now returning in the form of various applications. Home and desktop machines now can collaborate without any predefined infrastructure and form collaborative groups, share files, computing power, act as distributed search engines, etc. Privacy is also benefiting from this change, as the Peer to Peer mode of communication can be made difficult to spy on - one example of this is the Freenet project [26]. It can be referred to as the secure, anonymous Internet within the Internet. Participating nodes donate a part of their cpu, bandwidth and storage resources and these form a kind of virtual web, where no content can be traced back to its origin or forcefully removed.

It is important to note, that filesharing networks have given P2P a form of notoriety - due to the common association between Peer to Peer, filesharing and copyright infringement. Another reason for this is that as mentioned in the previous paragraph, networks were being built up with the strong client - server data flow model in mind. Peer to peer changes all that and equalizes the upload and download flows, sometimes saturating the asymmetric Internet connections. The concept, however, is much larger than filesharing, and a lot of filesharing networks are used for lawful purposes. One example of filesharing that is mostly used for good purposes is BitTorrent [27]. The idea behind BitTorrent is to utilize the Peer to Peer principle to help with distributing large files across the Internet. The first user in the network hosts the *seed* - the original file. New nodes connect to the first user on a peer to peer basis and download portions of the file becoming servers of the portion of the file they already have themselves. In this manner, the more users are interested in acquiring the large file, the faster the process of downloading and sharing takes place. The protocol does not include search features, and the peer to peer networks are created solely for the purpose of easing the load on a server - as users in the BitTorrent association download portions of the file from other users, the usage of the bandwidth on the originating server is minimal. To give a few examples of the protocol being used for lawful purposes:

1. Knoppix distributing ISO images of their Linux operating system [29]
2. Machinima* site "Red vs Blue" distributes episodes of their own production [28]

> *Machinima - The term concerns the rendering of computer-generated imagery (CGI) with ordinary PCs and the 3D engines of first person shooter video games in real-time (on the computer of either the creator or the viewer) rather than offline with huge render farms. - Wikipedia.org

Peer to peer also extends beyond the concept of filesharing networks. Many such uses actually preceded filesharing entirely. Two notable examples are the Usenet and DNS which, while not pure Peer to Peer, can be considered the grandparents of today's Peer to Peer systems.

Usenet started around 1979 as a simple protocol based on the *Unix to Unix copy protocol*, or UUCP. The idea behind it is that a Unix machine would dial another Unix machine, exchange files and disconnect. This is extended to exchanging data such as email files and article postings. The dial-in method is later replaced by the TCP/IP transport layer, and the protocol modeled after UUCP is called *Network News Transport Protocol* [32] or NNTP and is in widespread use today. No clear-cut hierarchy exists within the Usenet world, except for the hierarchy in the article system itself. A server joins the article exchange by establishing a connection with an already participating server and exchanging articles on a regular basis. A user then can read the articles in the Usenet system by contacting one of the participating servers - each of the servers has access and can carry the full Usenet traffic, although some choose to carry only a part of it in order to reduce the load. In addition to the article distribution between servers being done on a P2P basis, the article hierarchy itself lacks any centralized control. The whole process of adding and removing newsgroups is democratic with one notable exception - the alt.* branch, in which a single user can create a newsgroup by himself. Usenet contains a simple Peer to Peer flood avoidance system - each message carries a Path header, which contains the names of the servers it has passed through. Each server checks the header for their own name and will not attempt to retransmit a message it has already transmitted once on a particular link.

DNS is a hybrid Peer to Peer and hierarchical system. It evolved as a solution to the problem that plagued the early Internet - name resolution is originally done via the means of a hosts.txt that contained the names and corresponding IP addresses of all servers on the Net. As the classic way of distributing the updated file is to ftp it to all the machines on the Net, it quickly turned out that with the growing number of machines, the task was becoming simply unmanageable. This called for a solution that would mean constructing an online distributed database that would respond to DNS queries. The database had to be distributed not centralized because as the system grew, the flux of information and queries would simply overwhelm any single machine. The obvious solution is to use the Peer to Peer model, where as the number of users grows, so does the DNS system. Each node would act as both a client and a server - taking queried and propagating them (querying other nodes) if it couldn't answer itself. The hierarchy is necessary to ensure order in the network - the way URL's are constructed dictates the hierarchy in the DNS system. The highest order DNS nodes handle the top-level domains - such as .com .org, the national top-level domains such as .dk .pl .es etc. The lower order domains follow lower in the hierarchy - like the amazon.com subdomain or the yahoo.com. The hierarchy can be dynamically extended to cover any depth. As today's Internet shows, the system still works remarkably well - having scaled from thousands to hundreds of millions of hosts over the course of 21 years [20]. This proves the viability and power behind the concept of Peer to Peer.

3.2 Ad Hoc networks

As mentioned in section 1.2 on page 12 an ad-hoc network can be defined as a network that does not require existing infrastructure in order to work. These are well suited to needs of a temporary nature or ones that require networks to be formed in the field, on the fly.

Body Area Networks are an application of ad-hoc networks that interconnects wearable digital platforms and makes them available to the user. One example could be a headset and watch displays interconnecting with the user's mobile phone and PDA. The devices then may cooperate - e.g. an audible sound may be played during a mobile phone conversation announcing that there is a scheduled meeting on the PDA.

Personal Area Networks interconnect Body Area Networks of different people in vicinity and allow them to interact with the environment. The applications for this may include interactions with:

- Information kiosks
- Electronic news stands
- Internet access points
- Building Area Networks and others

Users will be able to exchange electronic calling cards, files, etc.

Other scenarios where Ad Hoc networks may be used are for example:

- Emergency services and disaster recovery
- Medical institutions
- Battlefield networking
- Sensor dust

Hospitals and emergency services can make use of as hoc technology by e.g. providing each patient with a chip to replace the patient's medical history card. Furthermore devices monitoring the patient's condition can interface to the hospital network and keep track of the vital statistics, alerting doctors to any sudden changes.

Disaster recovery situations are ones where Ad hoc technology is already used. In such situations effective communication and coordination is essential. The TErrestrial Trunked RAdio (TETRA) [33] communications system provides reliable wireless communications for emergency services around the world and in any location. Furthermore teams may use Wireless LAN technology to establish command locations and interface them to mobile teams via wireless interfaces. Victims in numerous disasters managed to notify the search and rescue teams by using their GSM phones - the teams can then triangulate the signal and coordinate efforts to help the survivors.

Battlefield networking is also a field, where instant information and coordination is essential to success and minimizing casualties on both sides. The soldier will be equipped with a HUD and wearable computer providing a tactical uplink to the command center. The soldier will transmit his location and estimates on encountered resistance, and will in turn receive similar information gathered and processed from other soldiers in the vicinity. Using bulky and heavy satellite transmitters for this purpose is not an option, therefore a secure wireless Ad hoc network is established on the battlefield linking the soldiers together.

Sensor dust is a novel concept, where multiple small and simple sensor devices are deployed over a large area. These then establish an Ad Hoc network using low power short range transmitters and start sending sensor and location information over the network. Such smart dust sensor networks can be deployed in hostile environments, as such a network can still function if a large percentage of devices malfunctions.

3.2.1 Ad Hoc network technology

Ad-hoc networks are usually defined as a temporary association of nodes, serving a specific function. Ad-hoc networks tend to be very dynamic, and varying in size and complexity. The most widely used and largest ad-hoc networks are the cellular telephony networks. These started as analogue circuit switched AMPS/TACS networks, then evolved to 2G GSM [5](900MHz/1800MHz European)/PCS [4](800MHz/1900MHz US band used for GSM/CDMAOne/iDEN/D-AMPS) digital circuit switched networks. The newest addition to the mobile phone family is the 3G digital packet switched mobile telephony standard - WCDMA (UMTS [8] in Europe and FOMA/J-Phone in Japan) and CDMA2000 [9] in the US.

The smaller wireless computer networking technologies are mainly IrDA [13], Bluetooth [11] and 802.11a-g standards (WLAN [10] and others). One or more of these technologies are embedded in today's PDA's and portable computers. Furthermore watches and other portable devices are beginning to have support for wireless networking technologies.

Figure 3.2 shows a comparison between wireless infrastructure and ad-hoc environments. Wireless infrastructure requires the presence of Base Transmitter Stations, while ad-hoc requires no pre-existing infrastructure.

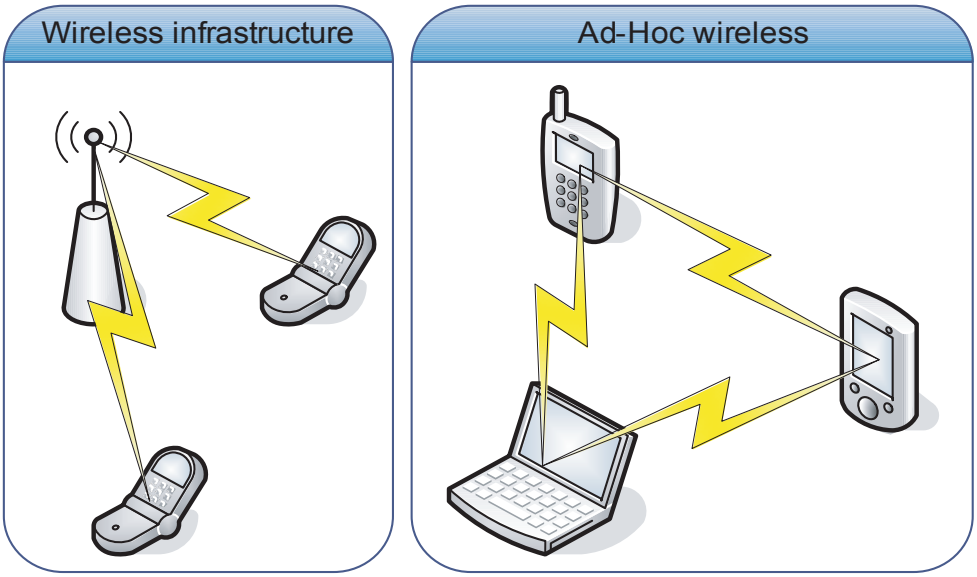

Figure 3.2: Wireless infrastructure and Ad-Hoc

3.2.2 Wireless Peer to Peer

In today's world, mobile devices have much greater capabilities than they used to - palmtops now come with 400MHz CPU's, 256MB of RAM and large storage, making them a match for desktop

computers from a few years ago. If this trend continues, then the complexity of applications that are available on such devices are more than likely to match desktop, wired computers. The mobility and ability to form ad-hoc associations is an added bonus that effectively extends the scope of such applications. The most basic example is that a single device needs not have a direct connection to every other mobile node in its vicinity. As long as the low power transceiver can reach nodes that in turn can forward packets to further nodes, the reach of an individual mobile node is greatly extended with power requirements only slightly increased - due to forwarding of other nodes' packets.

3.3 Summary

This chapter introduces the Peer to Peer concept in detail. A brief history of P2P is presented. The very concept of the Internet is based on P2P - all machines were meant to serve as both clients and servers at the same time. Commercialization of the Internet and the introduction of ADSL changed this approach, and the Internet began to follow the client-server concept. However, with the growth of number of clients, the required server architectures are no longer cost-effective. The solution is to go back to the P2P concept, where each client would also share part of the service provision. File sharing networks such as Napster and Kazaa evolved out of this concept, and became synonymous with the otherwise general term Peer to Peer. A particularly interesting application of P2P is introduced in section 3.2 - Ad-Hoc networks. In infrastructure-less situations wireless devices can interact using P2P to create dynamic networks. Emergency services and battlefield communication are mentioned as primary applications. The probabilistic trust valuation concept introduced in the previous chapter is applicable in such environments. Chapter 4 will discuss the concept in detail.

Chapter 4

Peer to Peer PKI: Probabilistic confidence valuation logic

4.1 Introduction

As mentioned in the introduction and stated in the two previous chapters, the goal of this report is to extend the work done in the field of infrastructure-less PKIs. The paper by Ueli Maurer "Modelling a Public-Key Infrastructure" is particularly important for this report, as many of the theoretical guidelines presented there are used as basis for the practical implementation presented in this report.

A PKI will be thought of here as a heterogeneous peer-to-peer network of nodes. The implementation focuses on the probabilistic public-key confidence valuation model, which is providing a means of performing two basic functions that are part of any PKI:

- Retrieving and verifying the authenticity of user Bob's public-key
- The ability to contribute to the distributed PKI by providing statements attesting to the authenticity of other user's public-keys

The algorithm focuses in particular on user queries and the ability to retrieve certificates and statements about certificates from the network. Aspects of storage of certificates and statements on nodes are also addresses. Certificates can be stored using a number of methods - the hash tables in particular was used in the implementation presented in this report.

Drawing motivation from previous work such as PGP by Philip Zimmerman [23] and the aforementioned "Modelling a PKI Infrastructure" by Ueli Maurer [19] the implementation effectively uses multiple certification paths through a network. A confidence parameter is introduced, which is assigned to each of the individual statements comprising the certification path. The paths are then aggregated using probabilistic argumentation, and an aggregate confidence parameter is obtained. This parameter can then be used to make threshold-logic decisions about whether to allow a Node to perform a certain action or not.

Similarly to other publications [35] in the field of cryptography, the examples and explanations will be given using fictitious `Actors`. The actors involved will be as follows:

Name	Node ID	Role
Alice	Node 0	Wants to obtain authenticity information about Bob
Bob	Node 3 or 4	Passive
Carol	Node 1	Responds to Alice's queries
Dave	Node 2	Responds to Alice's queries
Eve	Node 3	Responds to Alice's queries

It is important to note, that although the actors are referred to as Alice, Bob, Carol, Dave, Eve, they need not represent actual users. They will most likely be the access-control encryption mechanism on a user device, a peer-to-peer application or similar.

4.2 Public-key certification mechanism

Alice can only be certain about Bob's public-key when there have been specific conditions met. Alice should have received Bob's certificate - containing his public key and a signed statement that the public-key belongs to Bob. Furthermore Alice has to be convinced of the certificate's *authenticity*. Alice has to also *trust* the entity that signed Bob's certificate.

Propagation of authenticity of public-keys is done in a straight-forward way. The method is already used in PGP's "Web of trust", where a certification chain must exist from Alice to Bob, so that Alice may trust Bob's certificate. PGP's "Web of trust" means in this context a web of certificates, as trust is treated as a separate issue.

An example of a certification chain is shown on figure 4.1 below:

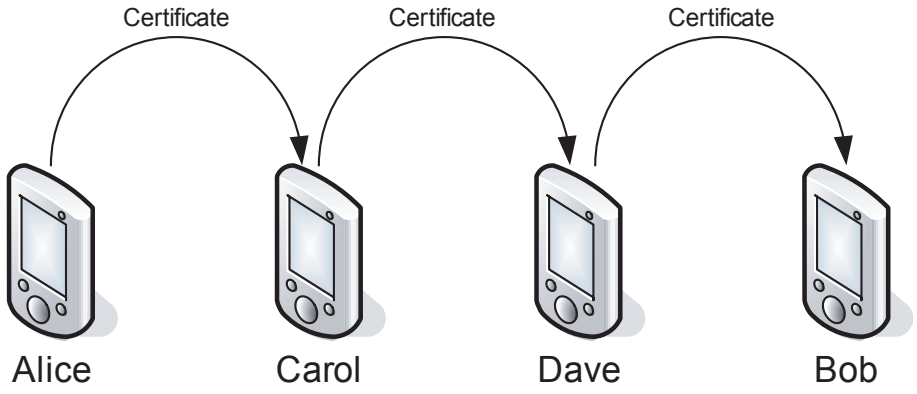

Figure 4.1: Certification chain

The certification chain shows a situation, where Alice can be assured of the authenticity of Bob's public key by verifying a certification chain. Carol's certificate is signed with Alice's signature, Dave's certificate is signed by Carol, and Bob's certificate is signed by Dave. Therefore a chain of valid certificate signatures exists, so that Alice can be assured of the validity of Bob's certificate and in turn of the authenticity of his public key.

As an extension to the certification chain, and as suggested in [19], trust and recommendations are introduced. A recommendation can be thought of a signed statement testifying to the trustworthiness of a particular entity. This can be compared to real life, where recommendations play a very important role. For example in the recruitment process for a job, a prospective employer will be looking for good references from former employers of the person he wishes to

hire. Such recommendations are considered explicit, as they refer to the person in particular. One may also consider references that are implicit - such as that a former Rangers operative would be perfect for a job at a security company.

In confidence parameter valuation, Alice has to trust a particular node if she is to accept a signed certificate or signature from that node and consider it as valid. The relation that Alice is ready to accept signed statements from another node will be referred to as trust. The relation that a particular node can recommend another node as trustworthy to Alice will be referred to as a recommendation of level 2. The recommendation that a particular node can recommend another node as trustworthy to Alice will be referred to as a recommendation of level 3 and so on.

Recommendations play an important role, but can also be considered sensitive information. For example, it may not be a good idea to let Alice tell the nodes around her whether she trusts them or not.

An example of a certification chain with trust statements is shown in figure 4.2

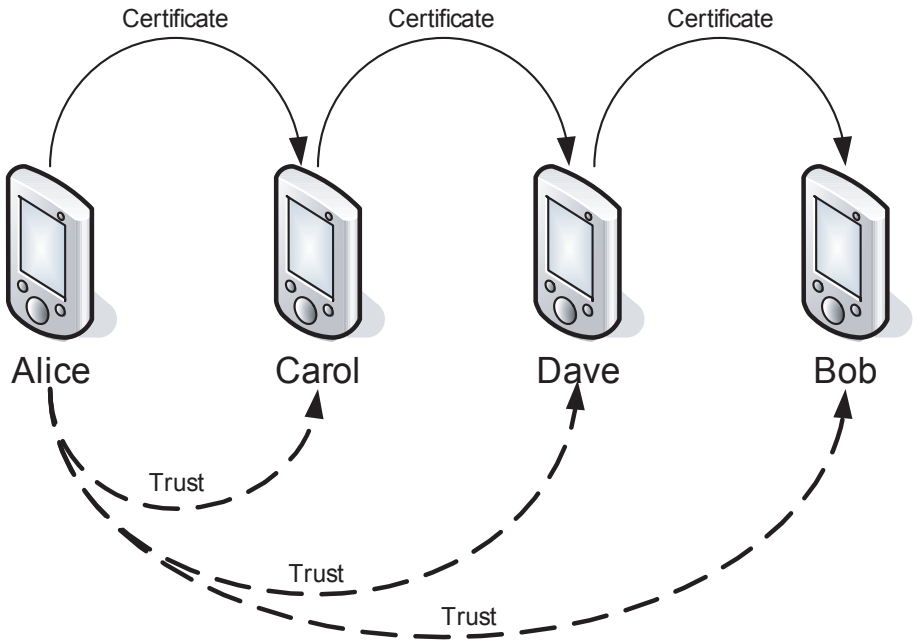

Figure 4.2: Certification chain with trust

It is easy to see the certification chain from Alice to Bob, however the trust statements require a few words of explanation. Every node in the certification chain except Alice and Bob needs to be trusted. This is due to the fact that Alice trusts herself, and she need not consider Bob as trustworthy for issuing signed statements about other nodes' certificates nor their trustworthiness. Alice is only interested in verifying the authenticity of Bob's public-key - the confidence in the statement $Aut_{A,B}$. Trust in Bob - $Trust_{A,B}$ is optional.

4.3 Syntax

4.3.1 Statements

The term `point of view` used here will refer to a set of information a node has obtained about the network. This will be the certifications and trust of the node itself in other nodes currently

present in the network, as well as certifications and recommendations of other nodes.

In describing the state of the network, and the points of views of nodes, the following kinds of statements are used:

- Authenticity of public keys. $Aut_{A,C}$ pronounced as "Belief in the authenticity of Node C's public key in view of Alice". Graphically this statement is represented as an edge from A to C.
- Trust $Trust_{A,C,1}$ pronounced as "Belief that Node X is trustworthy for signing certificates in view of Alice". A trust of higher level $Trust_{A,C,i}$ is pronounced as "Belief that Node X is trustworthy for issuing recommendations of level i-1 in view of Alice". Graphically this is represented as a dashed edge from A to C.
- Certificates $Cert_{B,C}$ pronounced as "Belief in authenticity of Node C's public key in view of Node B". This basically means that Alice has obtained C's public key signed by B.
- Recommendations $Rec_{B,C,i}$ pronounced as "Belief that Node C is trustworthy for issuing recommendations of level i-1 in view of Node B". This basically means that Alice has obtained a statement of Trust of level i from Node B for Node C, signed by Node B.

Therefore, the example from figure 4.2 can be redrawn as shown in figure 4.3

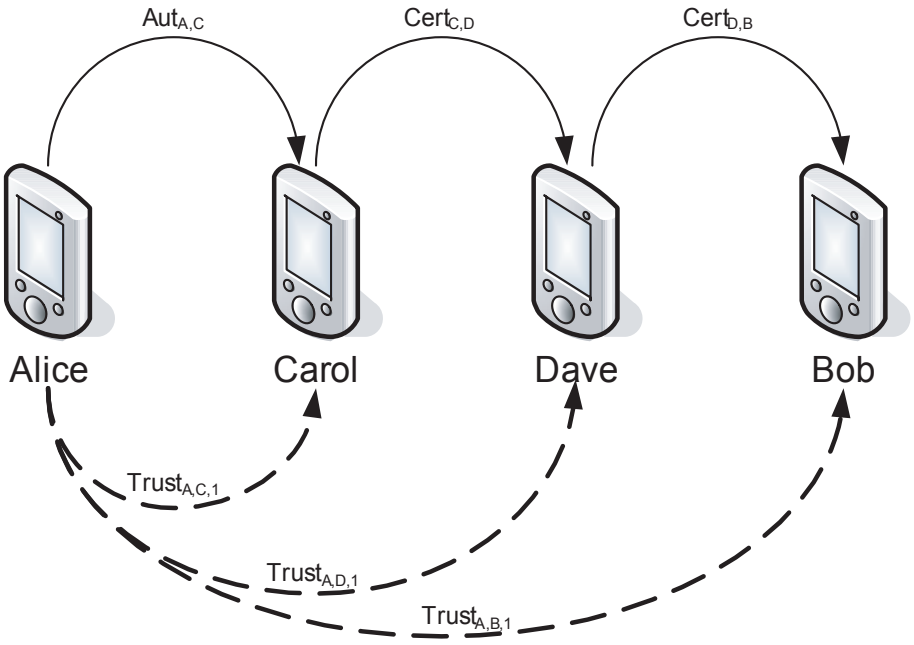

Figure 4.3: Certification chain with trust

It is important to note, that when describing the network, the meaning of the statements becomes relative. E.g. in the network shown in figure 4.3, from Alice's point of view the network looks like this:

$\{Aut_{A,C}, Cert_{C,D}, Cert_{D,B}, Trust_{A,C,1}, Trust_{A,D,1}, Trust_{A,B,1}\}$

However, from the point of view of Dave, the network looks like this:

$\{Cert_{A,C}, Cert_{C,D}, Aut_{D,B}, Rec_{A,C,1}, Rec_{A,D,1}, Rec_{A,B,1}\}$

Some of the Aut and $Cert$ statements have swapped place, while all $Trust_{A,..,1}$ statements have become $Rec_{A,..,1}$(recommendation that a given node is trustworthy for issuing signed statements).

4.4 Derivation rules

This section will introduce the derivation rules that can be used to derive new statements from a given view.

The two basic derivation rules that are used throughout this report are presented below:

$$\forall X, Y: \quad Aut_{A,X}, Trust_{A,X,1}, Cert_{X,Y} \implies Aut_{A,Y} \tag{4.1}$$

The above states that given Alice's belief in the authenticity of X's public key, and given Alice's belief in trustworthiness of X to sign certificates, should X sign Y's public key and give it to Alice, Alice will draw the conclusion that Y's public-key is authentic.

$$\forall X, Y, i \geq 1: \quad Aut_{A,X}, Trust_{A,X,i+1}, Rec_{X,Y,i} \implies Trust_{A,Y,i} \tag{4.2}$$

4.5 Probabilistic model

Equation 4.2 is the derivation statement used when reasoning about trust. The first two terms on the left side of the equation say that Alice has to believe in the authenticity of X's public key and trust X is trustworthy for issuing recommendations of level i. Should X sign a recommendation statement of level i for node Y, Alice will draw the conclusion that Node Y is trustworthy for issuing recommendations of level i-1.

Applying the first rule, it is easy to derive new statements from the network given in figure 4.3.

$$Aut_{A,C}, Trust_{A,C,1}, Cert_{C,D} \implies Aut_{A,D}$$
$$Aut_{A,D}, Trust_{A,D,1}, Cert_{D,B} \implies Aut_{A,B}$$

Therefore it was possible to derive the desired statement $Aut_{A,B}$. As mentioned before, the trust statement $Trust_{A,B,1}$ was not required to complete the derivation.

Figure 4.4 shows a more complicated example, with higher level trust statements.

The derivation of statement $Aut_{A,B}$ is also quite straightforward, as long as we keep in mind that trust and recommendations of level i $TrustA, .., i/RecA, .., i$ imply trust and recommendations of lower levels. Therefore we can state:

$$Aut_{A,C}, Trust_{A,C,1}, Cert_{C,D} \implies Aut_{A,D}$$
$$Aut_{A,C}, Trust_{A,C,3}, Rec_{C,D,2} \implies Trust_{A,D,2}$$
$$Aut_{A,D}, Trust_{A,D,1}, Cert_{D,B} \implies Aut_{A,B}$$

and optionally, the trust for Bob can also be derived:

$$Aut_{A,D}, Trust_{A,D,2}, RecD, B, 1 \implies Trust_{A,B,1}$$

The two examples also show that if the view does not contain any recommendations of level higher than 1, then Alice has to trust every intermediate node in the certification path.

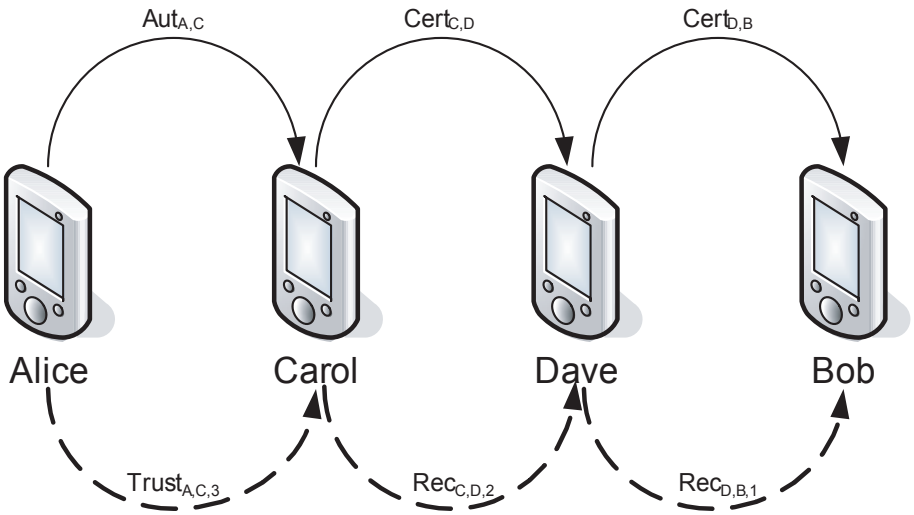

Figure 4.4: Certification chain with higher level trust

4.6 Probabilistic confidence valuation

The problem of computing the confidence value for a statement $Aut_{A,B}$ can be solved by explicitly calculating all the views V that contain the statements necessary to derive $Aut_{A,B}$ and summing the probabilities of the views. This is, however, akin to an exhaustive search and cannot be applied effectively in networks consisting of a large number of nodes.

Ueli Maurer proposes [19] a more efficient approach that takes the minimal subsets $V_1, V_2, \ldots V_p$ that lead to derivation of $Aut_{A,B}$. Then the statement $Aut_{A,B}$ can be derived from any subset of S_a that contains one or more of the minimal subsets. The probability that the statement $Aut_{A,B}$ can be derived will be referred to as the confidence parameter $\text{conf}(Aut_{A,B})$.

Therefore the probability of $Aut_{A,B}$ can be expressed as follows:

$$\text{conf}(Aut_{A,B}) = P(\bigvee_{i-1}^{p}(V_i \subseteq View_A)) \tag{4.3}$$

The above formula is expanded according to the inclusion-exclusion principle which states that [34]:
Given a $p - system$[1] $S_a = \{S_i\}_{i=1}^{p}$ consisting of sets $S_1, \ldots, S_p$

$$\begin{aligned} P(S_1 \cup S_2 \cup \ldots \cup S_p) = &\sum_{1 \le i \le p} P(S_i) - \sum_{1 \le i_1 \le i_2 \le p} P(S_{i1} \cap S_{i2}) \\ &+ \sum_{1 \le i_1 \le i_2 \le i_3 \le p} P(S_{i1} \cap S_{i2} \cap S_{i3}) \\ &- \ldots + (-1)^{p-1} P(S_1 \cap V_2 \cap S_3 \cap \ldots \cap V_p) \end{aligned} \tag{4.4}$$

We should also note that each view V_i is in fact a collection of statements S_i:

[1] $p - system$: a sequence of subsets of a set S. The subsets may be empty or have non-empty intersections.

$$V_i = \{S_1, S_2, \ldots, S_k\}_{1 \leq k \leq p} \tag{4.5}$$

The probability that a given subsequence $V_i \subseteq View_A$ is expressed as:

$$P(V_i \subseteq View_A) = P(S_1 \subseteq View_A) \cdot P(S_2 \subseteq View_A) \cdot \ldots \cdot P(S_k \subseteq View_A) \tag{4.6}$$

It is worth noting that

$$\begin{aligned} & P((V_i \subseteq View_A) \cap (V_j \subseteq View_A)) \\ &= (P(S_1 \subseteq View_A) \cdot P(S_2 \subseteq View_A) \cdot \ldots \cdot P(S_k \subseteq View_A)) \cap (P(S_1 \subseteq View_A) \cdot P(S_2 \subseteq View_A) \cdot \ldots \cdot P(S_l \subseteq View_A)) \\ &= P(S_1 \subseteq View_A) \cdot P(S_2 \subseteq View_A) \cdot \ldots \cdot P(S_k \subseteq View_A) \cdot P(S_1 \subseteq View_A) \cdot P(S_2 \subseteq View_A) \cdot \ldots \cdot P(S_l \subseteq View_A) \end{aligned} \tag{4.7}$$

And noting that if the views V_i, V_j contain intersecting statements $P(V_i \subseteq View_A) \cap P(V_i \subseteq View_A) = P(V_i \subseteq View_A)$, the confidence value $conf(Aut_{A,B})$ can then be expressed as:

$$\begin{aligned} \text{conf}(Aut_{A,B}) = & \sum_{1 \leq i \leq p} P(V_i \subseteq View_A) \\ & - \sum_{1 \leq i_1 \leq i_2 \leq p} P((V_{i1} \cup V_{i2}) \subseteq View_A) \\ & + \sum_{1 \leq i_1 \leq i_2 \leq i_3 \leq p} P((V_{i1} \cup V_{i2} \cup V_{i3}) \subseteq View_A) \\ & - \ldots + (-1)^{p-1} P((V_1 \cup V_2 \cup V_3 \cup \ldots \cup V_p) \subseteq View_A) \end{aligned} \tag{4.8}$$

4.7 Summary

Probabilistic confidence valuation is inspired by the PGP Web of Trust. A certification chain used to verify the authenticity of public-keys is extended with the concept of trust. Alice has to be aware of a certification chain to Bob, and she has to trust every intermediate node either directly or through a higher level trust path. The concepts of authenticity and trust can be used to capture a number of interrelations in networks of nodes. Basic syntax is introduced along with derivation rules that can be used to reason about authenticity and trust.

Chapter 5

Design

This chapter introduces the design of the software prototpe used to verify the feasibility of the probabilistic confidence valuation concept. Chapter 3.2 introduces the theoretical basis for confidence parameter calculation. This chapter builds on the theory and presents a number of algorithms that can be used to implement a software prototype.

5.1 Algorithm

Based on the theory discussed in section 4 the algorithms that need to be developed are:

- Syntax handling
- Finding a certification path
- Finding a trust path
- Calculating the confidence value (Inclusion-Exclusion principle)
- Peer to peer harness

5.1.1 Finding a certification path

The first step to acknowledging a certificate of node B in the network is finding a chain of certificates that links A and B. The derivation rules contained in Ueli Maurers paper are as follow:

$$\forall X, Y: \quad Aut_{A,X}, Trust_{A,X,1}, Cert_{X,Y} \implies Aut_{A,Y} \tag{5.1}$$

$$\forall X, Y, i \geq 1: \quad Aut_{A,X}, Trust_{A,X,i+1}, Rec_{X,Y,i} \implies Trust_{A,Y,i} \tag{5.2}$$

Equation (5.1) is of special importance here. It implies, that to make a statement $Aut_{A,B}$ there needs to exist, at a minimum, a chain of certificates e.g:

$$\{Aut_{A,X}, Cert_{X,Y}, CertY, Z, CertZ, B\} \tag{5.3}$$

Therefore an algorithm for finding certification paths through a network of nodes needs to be designed. The natural candidate is the Bellman-Ford algorithm. The algorithm operates on an adjacency matrix representation of a network of nodes.

The algorithm can be written using the following pseudo code:

```
BELLMAN-FORD(V[])
1 Initialize d[] to 65535 and pi[] to -1
2 d[0]=0
3 for i=1 to length of V[] - 1
4  for each edge (u,v) in V[]
5  RELAX(u,v,w)

RELAX(u,v,w)
1 if d[v]> d[u] + w(u,v)
2  then d[v] = d[u] + w(u,v)
3 pi[v] = u
```

The algorithm operates on the assumption that the shortest path to a node is the collection of locally shortest paths. Therefore each iteration it passes through all the edges in the network, *relaxing* them - checking if the value at the destination vertex is greater than the value at the source vertex plus the weight of the path. The iteration is repeated n times, where n is the number of nodes in the network.

The Bellman-Ford algorithm alone is insufficient for the task of locating independent certification paths. Ideally we would like to find all possible certification paths so as to maximize the obtained confidence value. I have developed a recursive solution building on the B-F algorithm:

```
CERTPATHS(Sa)
1 Initialize Results[]
2 Path=BellmanFord(Sa)
3 If Path is not null, and it is not in Results[]
4 store Path in results
5 for integer i=1 to number of statements in Path
6 remove statement i from Sa
7 CERTPATHS(Sa)
```

This algorithm will obtain the shortest path using the Bellman-Ford algorithm, then iterate through the certification statements comprising the path and remove them one at a time, calling `BellmanFord(Path)` recursively with each iteration. As the optimal path is 'broken' the Bellman-Ford algorithm will find the next best path, and the process continues recursively until all paths are found. This algorithm is much better than an exhaustive search, however it can be considered quite aggressive. It will work very well for small networks, where it will make best use of the available resources. For larger views, however, genetic algorithms could be considered to find unique certification paths. This is, however, outside the scope of this report.

5.1.2 Finding a trust path

Going back to Ueli Maurers derivation rules:

$$\forall X, Y: \quad Aut_{A,X}, Trust_{A,X,1}, Cert_{X,Y} \implies Aut_{A,Y} \tag{5.4}$$

$$\forall X, Y, i \geq 1: \quad Aut_{A,X}, Trust_{A,X,i+1}, Rec_{X,Y,i} \implies Trust_{A,Y,i} \tag{5.5}$$

After finding a certification path, the next step is to ensure that each step along the way is trusted by the source node. The algorithm developed here uses a modified version of the Bellman-Ford algorithm where the additive relaxation of the edges is replaced by multiplicative relaxation.

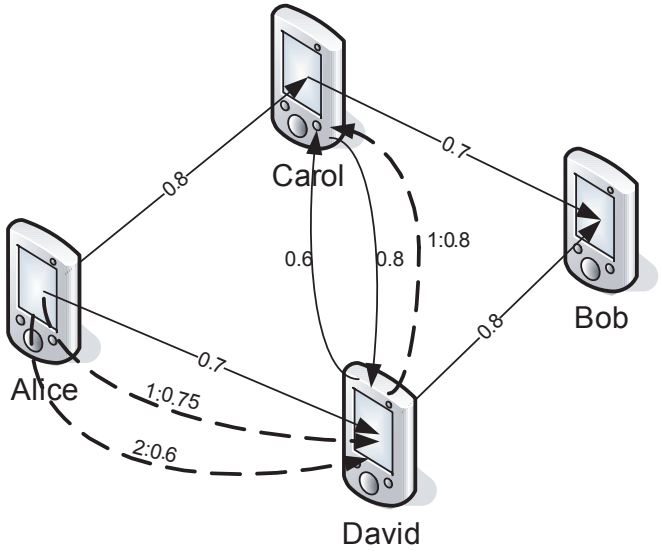

Figure 5.1: Example network

```
BELLMAN-FORD(V[])
1 Initialize dd[] to 0.0 and pi[] to -1
2 dd[0]=1.0
3 for i = 1 to length of V[] - 1
4  for each edge (u,v) from V[]
5  RELAX(u,v,w)

RELAX(u,v,w)
1 if d[v]< d[u] * w(u,v)
2  then d[v] = d[u] * w(u,v)
3 pi[v] = u
```

There is, however a problem with this approach which can be demonstrated by looking at the example in figure 5.1. The network is small, comprising of 4 nodes, 6 certification and 4 trust edges. By applying the algorithm described in section 5.1.1 it is easily found that the possible certification paths through the network are:

$$A - X - B$$
$$A - Y - B$$
$$A - X - Y - B$$
$$A - Y - X - B$$

These certification paths are represented by the following certification statements:

$$V_1 = \{Aut_{A,X}, Cert_{X,B}\}$$
$$V_2 = \{Aut_{A,Y}, Cert_{Y,B}\}$$
$$V_3 = \{Aut_{A,X}, Cert_{X,Y}, Cert_{Y,B}\}$$
$$V_4 = \{Aut_{A,X}, Cert_{Y,X}, Cert_{X,B}\}$$

The next step calls for finding the trust statements to support the given certification paths. The paths are found using the modified Bellman-Ford algorithm for each node in the certification path except the source node A and the destination node B

$$\begin{aligned}
V_1 &= \{Aut_{A,X}, Cert_{X,B}\} + \{Trust_{A,X,1}\} = \{Aut_{A,X}, Cert_{X,B}, Trust_{A,X,1}\} \\
V_2 &= \{Aut_{A,X}, Cert_{X,B}\} + \{Trust_{A,Y,1}\} = \{Aut_{A,X}, Cert_{X,B}, Trust_{A,Y,1}\} \\
V_3 &= \{Aut_{A,X}, Cert_{X,Y}, Cert_{Y,B}\} + \{Trust_{A,X,1}, Trust_{A,Y,1}\} \\
&= \{Aut_{A,X}, Cert_{X,Y}, Cert_{Y,B}, Trust_{A,X,1}, Trust_{A,Y,1}\} \\
V_4 &= \{Aut_{A,X}, Cert_{Y,X}, Cert_{X,B}\} + \{Trust_{A,Y,1}, Trust_{A,X,1}\} \\
&= \{Aut_{A,X}, Cert_{Y,X}, Cert_{X,B}, Trust_{A,Y,1}, Trust_{A,X,1}\}
\end{aligned}$$

In the above the probabilities that the trust paths are contained in $View_A$ are as follows:

$$\begin{aligned}
V_1 &: P(Trust_{A,X,1}) = 0.7 \\
V_2 &: P(Trust_{A,Y,1}) = 0.85 \\
V_3 &: P(Trust_{A,X,1}) \cdot P(Trust_{A,Y,1}) = 0.7 \cdot 0.85 = 0.595 \\
V_4 &: P(Trust_{A,Y,1}) \cdot P(Trust_{A,X,1}) = 0.85 \cdot 0.7 = 0.595
\end{aligned}$$

I will now suggest an alternative trust path for views V_3 and V_4, V_3 and V_4 become:

$$\begin{aligned}
V_3 &= \{Aut_{A,X}, Cert_{X,Y}, Cert_{Y,B}, Trust_{A,Y,2}, Trust_{A,Y,1}, Rec_{Y,X,1}\} \\
V_4 &= \{Aut_{A,X}, Cert_{Y,X}, Cert_{X,B}, Trust_{A,Y,2}, Trust_{A,Y,1}, Rec_{Y,X,1}\}
\end{aligned}$$

Due to the fact that the second level trust $Trust_{A,Y,2}$ implies the existence of $Trust_{A,Y,1}$ we can say that:

$$P(Trust_{A,Y,2} \subseteq View_A) \cdot P(Trust_{A,Y,1} \subseteq View_A) = P(Trust_{A,Y,2} \subseteq View_A)$$

Given the above, the probability that the trust paths are contained in $View_A$ are now:

$$\begin{aligned}
V_3 &: P(Trust_{A,Y,1}) \cdot P(Trust_{A,Y,2}) \cdot P(Rec_{Y,X,1}) = 0.7 \cdot 0.9 = 0.63 \geq 0.595 \\
V_4 &: P(Trust_{A,Y,1}) \cdot P(Trust_{A,Y,2}) \cdot P(Rec_{Y,X,1}) = 0.7 \cdot 0.9 = 0.63 \geq 0.595
\end{aligned}$$

The example shows that due to second and higher level relationships it is not possible to optimally find global trust views as locally optimal ones do not necessarily form globally optimal ones. It is the belief of the author that an exhaustive search would be necessary to find globally optimal trust paths for trust levels higher than one. Therefore the algorithm developed will focus on trust level 1 and sub optimally on higher trust levels.

Developing an algorithm that can handle higher level trust optimally without performing an exhaustive search is considered suitable for further research. Genetic or neural network solutions could provide a way for solving this issue.

5.1.3 Calculating the confidence value

The problem of computing the confidence value for a statement $Aut_{A,B}$ can be solved by explicitly calculating all the views V that contain the statements necessary to derive $Aut_{A,B}$ and summing the probabilities of the views. This is, however, akin to an exhaustive search and cannot be applied effectively in networks consisting of a large number of nodes.

[19] proposes a more efficient approach that takes the minimal subsets $V_1, V_2, \ldots V_p$ that lead to derivation of $Aut_{A,B}$. Then the statement $Aut_{A,B}$ can be derived from any subset of S_a that contains one or more of the minimal subsets. Therefore the probability of $Aut_{A,B}$ can be expressed as in equation (4.8), repeated for convenience below:

$$\begin{aligned} conf(Aut_{A,B}) = \sum_{1 \le i \le p} & P(V_i \subseteq View_A) \\ & - \sum_{1 \le i_1 \le i_2 \le p} P((V_{i1} \cup V_{i2}) \subseteq View_A) \\ & + \sum_{1 \le i_1 \le i_2 \le i_3 \le p} P((V_{i1} \cup V_{i2} \cup V_{i3}) \subseteq View_A) \\ & - \ldots + (-1)^{p-1} P((V_1 \cup V_2 \cup V_3 \cup \ldots \cup V_p) \subseteq View_A) \end{aligned} \quad (5.6)$$

Mathematically speaking, the representation of $conf(Aut_{A,B})$ given above is very elegant, however programistically it presents a challenge. I have developed a recursive algorithm for solving equation (4.8) given an initial set of views V. The algorithm consists of two parts and is presented below in pseudo code:

The recursive part of the algorithm is `PERMUTATIONS( V, i, prefix )` and is defined as below:

```
PERMUTATIONS( V[], i, prefix ):
1 initiate Results[]
2 if i = 2
3 for integer j = 0 to length of V[] - 1
4 for integer k = j + 1 to length of V[]
5 store probability of ( V[j] + V[k] ) int Results[]
6 else
7 for integer j = 1 to length of V[] - i + 1
8 PERMUTATIONS(Views V[1..length], i-1, probability of View V[0])
```

The above algorithm takes as arguments a pair size i and set of views V. It then performs permutations as seen in equation (4.8) and stores the individual probabilities in the result array. For example the call

$$\texttt{PERMUTATIONS}(\{V_1, V_2, V_3\}, 2)$$

will yield

$$\{P((V_1 \cup V_2) \subseteq View_A), P((V_1 \cup V_3) \subseteq View_A), P((V_2 \cup V_3) \subseteq View_A)\} \quad (5.7)$$

in the result buffer.

The `CONFIDENCE(V[][])` algorithm uses `PERMUTATIONS( V[], i, prefix )` in the calculation of equation (4.8) as seen below:

```
CONFIDENCE(V[][])

1 Result = 0
2 For each view V in V[][]
3 Result = Result + probability of V[]
4
5 For integer i=2 to length of V[][] - 1
6 Array K[] = Permutations(V[],i,null)
7 For integer j=0 to length of K
8 if j is divisible by 2 then Result = Result + K[j]
9 else Result = Result - K[j]
```

The variable `Result` is used to store the final value of $conf(Aut_{A,B})$. First the variable is initialized to 0 and the probabilities of views $V_1 \ldots V_p$ are added together. Next the recursive algorithm `PERMUTATIONS(V[], i, prefix)` is called with increasing pair sizes from 2 to $lengthofV[] - 1$. The contents of the `PERMUTATIONS(V[], i, prefix)` result buffer are added or subtracted from `Result` depending on whether the pair size is divisible by 2.

This is a very concise and easy to understand way to solve equation (4.8), and can work for a theoretically unlimited number and size of views `V[]`. In practice it will be limited by available memory and maximum values of primitive data types used to store the variables.

5.1.4 Peer to Peer environment

The different algorithms presented in this chapter require a P2P test harness to provide a consistent binding. The solution that is proposed in this paper is designed for maximum clarity of concept. Communication between instances of network nodes is performed using native Java sockets. Queries are sent using XML and instances of the same class - network nodes - can act as peers - both clients and servers. A way onward would be to replace the socket-based Peer to Peer harness with a full P2P infrastructure such as JXTA. JXTA offers the following functionality:

- Service advertisement
- Service discovery
- logical P2P groups

Service advertisement and discovery is a useful feature that would allow for real-life implementations in the P2P environment. In the prototype presented in this paper, the nodes need to be made aware of the location of other nodes through user interaction. With JXTA the discovery and advertisement processes would be fully automated. The JXTA framework was not used in the prototype for a number of reasons. The first reason is clarity - the software prototype is to serve as a means for evaluating the feasibility of the probability valuation concept. Furthermore JXTA is often referred to as not fully mature - the system is still under development and not always stable. As this thesis was prepared in limited allotted time, the design decision was made not to use JXTA.

5.2 Summary

This chapter builds on the concepts introduced in the previous chapters, and presents the design of a software prototype that can be used to study the feasibility of the probabilistic trust valuation

concept. Algorithms and approaches are shown for syntax handling, finding a certification and trust paths and calculating the confidence value. A socket and XML based peer to peer test harness is used as a binding for the various modules of the prototype.

The design phase leads to several interesting conclusions about the feasibility of probabilistic confidence valuation. Finding trust paths for level 1 trust is relatively easy, however with the introduction of higher level statements, the problem becomes significantly harder. No algorithm except for exhaustive search can be suggested, as higher level trust introduces dependencies into the trust graph - the existence of a higher level trust statement implies the existence of a lower level trust statement.

The next chapter presents the actual implementation of the software prototype. The evaluation of the software is presented in chapter 7.

Chapter 6

Prototype Implementation

This section will present the actual implementation. It starts with an UML diagram presenting an overview of how the classes intertwine. The following sections will go through the class files one by one presenting which part of the design they realize, and presenting the implementation choices made. The code for the prototype is presented in Appendix A. A sample XML scenario file is presented in Appendix B.

6.1 UML Diagram

6.2 threaded_test_harness.java

instances with instantiated threads and waiting until they successfully complete their execution - using the *java.lang.Thread.join(long arg0)* method. Once the PNode threads have all finished, PNode[0].Sa contains a view with the statements retrieved from the network. This view may then be used to perform confidence parameter calculations using the *boolean p2p2ki.Paths.getResults(View V, int dest)* command.

6.3 Scenario_writer.java

Scenario files can be either created by altering the existing XML scenario files by hand or by using the supplied generator class. The sample code included is the code that is used to generate *Example4_7.xml*. By convention used in this implementation the instances of the *PNode* class are stored in a *Nodes[]* array. The size of the array should always be set to the number of nodes in the scenario. In our case this is:

```
PNode Nodes[] = new PNode[4];
```

Next, each node is configured using the getter and setter functions defined. The available settings are:

- `void p2p2ki.PNode.setConnectPorts(int[] ports)` - sets the array of ports that Alice will connect to to obtain the information about the other nodes in the network. Use of this setter should be restricted to Node 0 - Alice.

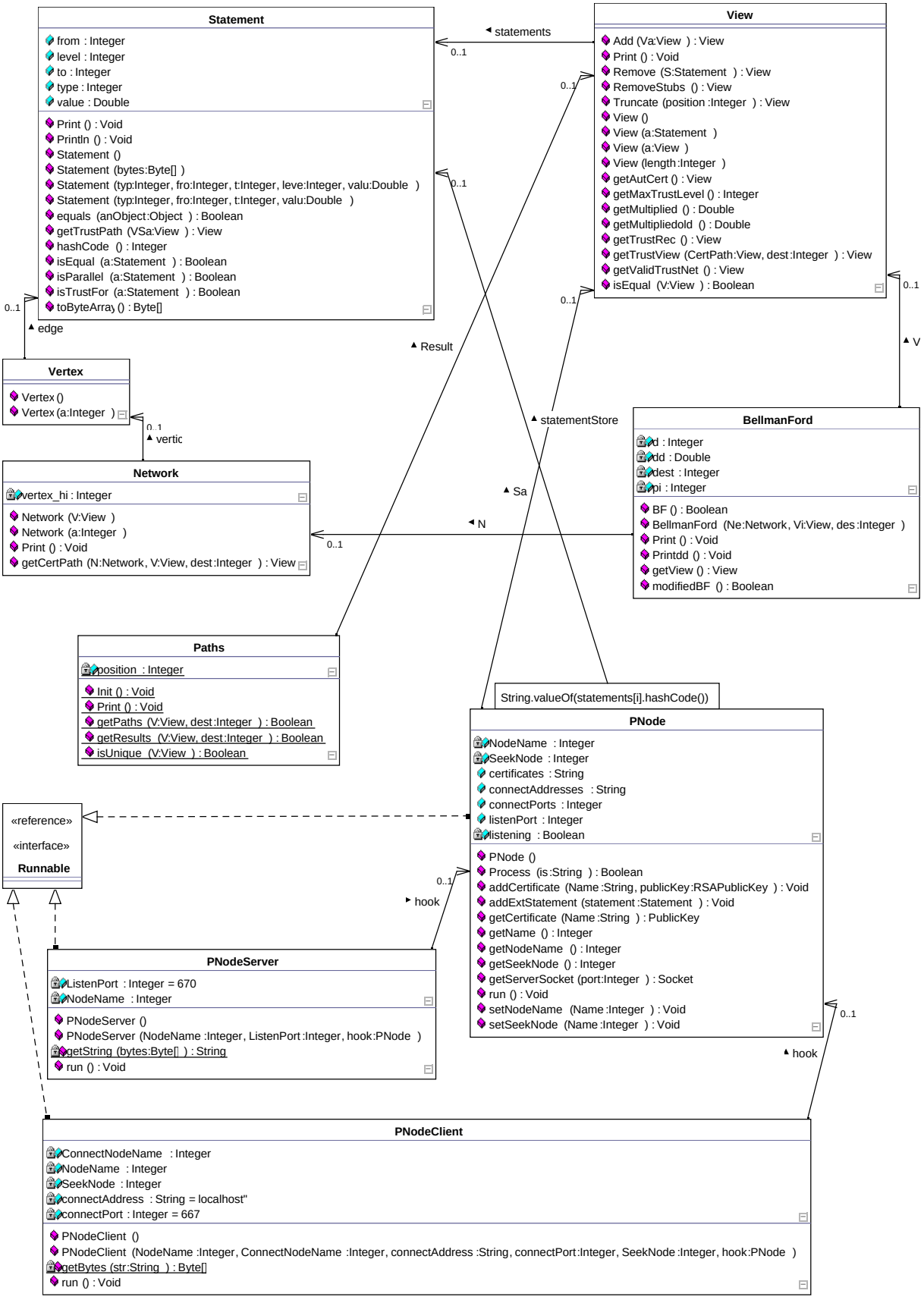

Figure 6.1: UML Diagram of the probabilistic public key valuation model implementation

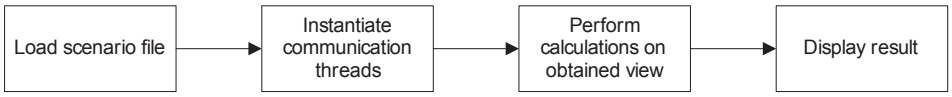

Figure 6.2: Threaded_test_harness.java flow

- void p2p2ki.PNode.setConnectAddresses(String[] addresses) - sets the array of URLs that Alice will connect to to obtain the information about the other nodes in the network. Used in companion with the *void p2p2ki.PNode.setConnectPorts(int[] ports)* method. Use of this setter should be limited to Node 0 - Alice.
- `void p2p2ki.PNode.setCertificates(String[] certnames)` - sets an array of paths that will be used to load public certificates into the nodes in the network. For simplicity the certificates are stored on a common directory on the hard drive and are read in at initialization time. After this time, the certificates are only exchanged using socket communication.
- `void p2p2ki.PNode.setStatementStore(Statement[] statements)` - sets an array of statements that the particular node is willing to make about the network surrounding it. The contents primarily reflect the contents of the certificate store - *Cert* statements from the point of view of Alice.
- `void p2p2ki.PNode.setNodeName(int Name)` - used to set the node name. Should be an integer starting at 0 for Alice.
- `void p2p2ki.PNode.setSeekNode(int Name)` - used to set the node being seeked out. Should only be used for Node 0 - Alice.

The configured network is saved into an XML scenario file with the filename provided as an argument. If the file already exists, the user will be presented with a prompt to either overwrite the existing file or exit.

6.4 PNode.java

The base class used when instantiating nodes in the network. Depending on settings this class will instantiate either PNodeClient or PNodeServer communication thread objects. PNode.java is the representation of a single node on the network and can play the role of both server and client - hence the peer to peer role.

Each object contains four hashtables:

- `public Hashtable AutStore` - stores the certificates read in at initialization time. The name AutStore implies that although from the point of view of Alice, the certificates are *Cert* statements, from the point of view of the particular node they are *Aut* statements - the node may play the role of Alice later on.
- `public Hashtable statementStore` - stores the statements the node can make about the network.
- `public Hashtable extAutStore` - stores the certificates Alice may collect from the network.
- `public Hashtable extStatementStore` - stores the statements Alice may collect from the network.

These hashtables are accessed using appropriate getter

- public String[] getCertificates()
- public Statement[] getStatementStore()

and setter functions:

- `public void setCertificates(String[])`
- `public void setStatementStore(Statement[])`
- `synchronized public void addCertificate(String, RSAPublicKey)`
- `synchronized public void addExtStatement(Statement)`

The last two setter functions are particularly interesting. When Alice instantiates a number of communication `PNodeClient` threads, they can securely store the obtained information in Alice using the `synchronized` mode setters. The `synchronized` classifier makes sure that the given method can only be held by one thread at a time. Failure to control this would result in a shared variable violation - corruption of stored information would result.

6.5 PNodeClient.java

This class is used by `PNode.java` as a communications thread object. Whenever Alice wishes to query another node about its view of the network, Alice can instantiate a `PNodeClient` object using the following statement:

```
p2p2ki.PNodeClient.PNodeClient(int NodeName, int ConnectNodeName, String connectAddress, int conn
```

The parameters are:

- `int NodeName` - the name of the `PNode` instantiating the client. The client identifies itself with the instantiating node.
- `int ConnectNodeName` - the name of the node to connect to
- `String connectAddress` - the URL of the node to connect to
- `int connectPort` - the port on the above URL to connect to
- `int SeekNode` - the name of the node being sought on the network
- `PNode hook` - a handle used to return information to the instantiating `PNode`

The instantiated `PNodeClient` will then attempt to connect to the given node and send a XML query of the form:

```
"<Query><DestNode Name="3"/></Query>\n"
```

where `"3"` stands for `int SeekNode`. The client then waits for a response on the same socket and processes the obtained certificates and statements storing them using `PNode hook` and the appropriate `synchronized` setter functions.

6.6 PNodeServer.java

This class is also used by `PNode.java` as a communications thread. Each node in the network can instantiate a listen server that waits and responds to XML queries from Alice. The instantiation is done using the following line:

```
p2p2ki.PNodeServer.PNodeServer(int NodeName, int ListenPort, PNode hook)
```

The parameters are:

- `int NodeName` - the name of the instantiating `PNode` class
- `int ListenPort` - the port at which the server thread should listen for queries
- `PNode hook` - `PNode` hook that is used when generating responses to queries

The instantaiated `PNodeServer` will stand by and listen for a XML query, respond on the same socket and exit.

6.7 Statement.java

The class is the core part of the syntax handling framework as shown in a section of the UML diagram - 6.1. The class is used for performing comparison operations on statements and as a basis for higher level syntax classes. The most important methods are:

- `public boolean isEqual(Statement a)` - returns true if the statement in the instance and the argument are equal
- `public int hashCode()` and `public boolean equals(Object anObject)` - required methods to be able to store `Statement` objects in a `Hashtable`. The `hashCode` method returns a string representing the contents of the statement, and the `equals` method returns true only if the argument object is is of type `Statement` and if it is equal to the statement stored in the called object.
- `public boolean isParallel(Statement a)` - returns true if the source and destination node IDs are the same. This method is used when checking for dependent statements (such as $Trust_{A,C,1}$ and $Trust_{A,C,2}$).

The class also contains getters and setters for the publicly available members. The duality of access (public members and getters/setters) is necessary to allow for saving scenario files using `XMLEncoder`. Each class saved using `XMLEncoder` must implement getters and setters for all the public members that are to be saved.

6.8 View.java

Second class in the core syntax handling framework. A view is essentially an array of statements. The methods available can be used to perform useful manipulation functions such as adding or removing a `Statement` from a `View`. The most important methods are as follows:

- `public View Add(View Va)` - Used to add two views together. In particular the second view may consist of a single statement, and then the function serves as adding a single `Statement` to a `View`.
- `public View getAutCert()` - Returns a view containing only *Aut* and *Cert* statements from the view stored in the called object. Used when locating a certification path.
- `public View getTrustRec()` - Returns a view containing only *Trust* and *Rec* statements from the view stored in the called object. Used when finding trust paths.
- `public View getValidTrustNet()` - When used on a view containing only *Trust* and *Rec* statements, the method will verify whether first level *Rec* statements are backed up by appropriate second or higher level *Trust* and *Rec* statements. Used when searching for a trust path through the network.
- `public View getTrustView(View CertPath, int dest)` - When used on a view containing a certification path, the method will find trust paths for each node in the network and return the aggregate trust view.

- `public double getMultiplied()` - Finds $P(V \subseteq View_A)$ - for independent statements, this is simply a multiplication of the individual probabilities $P(S_i \subseteq View_A)$. When dependent methods are encountered, they are not multiplied together - only the highest level statement is considered, the existence of lower level statements is implied.

As can be seen above, the class is heavily used in different parts of the implementation - when looking for certification and trust paths, and when calculating the confidence parameter.

6.9 permutations.java

The `permutations.java` class handles the core part of solving the inclusion-exclusion equation to obtain the confidence parameter as discussed in 5.1.3. The class is first called using:

```
static public double solveEquation(View V[])
```

where `View V[]` represents the array of minimal views from which $Aut_{A,B}$ can be derived. The method then calls the static

```
perm(View V[], int pair_size, View prefix)
```

method which recursively calculates the required combinations of views. The method is a direct implementation of the algorithm presented in section 5.1.3.

The main challenge when implementing this class was the choice of the size of the result buffer. The buffer is declared as:

```
public static DoubleBuffer ResultBuffer = DoubleBuffer.allocate(nnn)
```

When evaluating the performance of the implementation, it became apparent that the allocation and clearing of this buffer was taking a considerable amount of time - about 50% of execution time for a network of 4 nodes, 4 certification paths and `nnn`= 262144. `nnn` was chosen to be 1024, which is sufficient for medium and small networks. For larger networks, `nnn` should be increased for networks of more than 8 nodes and 12 certification paths.

The limit set is considered sufficient, as with Trust being a quickly fading resource, networks of more than 2-3 hops are unlikely to produce valid views from which $Aut_{A,B}$ can be derived.

6.10 Network.java

The `Network.java` class is a helper class designed to convert a view of a network to an adjacency matrix representation suitable for the BellmanFord shortest path algorithm. The design choice was made for performance reasons. The views are unsorted by design, and therefore each scan of the Bellman Ford algorithm would need to deduce the network structure from the view. Bellman Ford algorithm is likely to run faster on its native adjacency matrix representation - a structured network representation.

The conversion from view to adjacency matrix representation is done using the constructor:

```
public Network(View V)
```

Furthermore, the `Network.java` class contains the method used to find the certification path through the network:

`public View getCertPath(Network N, View V, int dest)`

where `Network N` is the adjacency matrix representation of `View V`. The latter contains only *Aut* and *Cert* statements. `int dest` contains the name of the destination node. The source node is assumed to be Node 0.

6.11 BellmanFord.java

The class `BellmanFord.java` implements two kinds of the Bellman Ford shortest path algorithm. It is first initialized with the constructor:

`public BellmanFord(Network Ne, View Vi, int des)`

where `Network Ne` is the adjacency matrix representation of `View Vi`. `int des` is the name of the destination node.

One of the two Bellman Ford algorithms is then called:

- Standard Bellman Ford - Implements the general additive, lower value is best Bellman Ford algorithm as discussed in 5.1.1. The result is stored in the `int d[]` minimum distance and `int pi[]` previous node arrays.
- Modified Bellman Ford - Implements the multiplicative, higher value is best Bellman Ford algorithm as discussed in 5.1.2. The result is stored in the `double dd[]` minimum distance and `int pi[]` previous node arrays.

The calling object then uses the `public View getView()` method. This method returns the resulting `int pi[]` array as a certification path, matching `int pi[]` against the stored `View Vi`.

Chapter 7

Evaluation

The probabilistic confidence valuation concept has been proven to be mostly feasible for implementation. A set of algorithms were developed and documented based on theoretical foundations. It was, however, noted that higher level trust paths require an exhaustive search to be performed. This presents an obstacle on the way to a practical implementation. A further research path would need to focus on developing an algorithm that could find higher level trust paths in less-than-exhaustive time.

7.1 Performance

The execution time of the implementation was tested on a number of small to medium networks from 4 to 8 nodes, with 2 to 12 certification paths. It was found that the algorithm scales reasonably well. For large networks, however, sensitivity analysis would be in order, as the algorithm attempts to find all available paths between source and destination nodes. This is extremely beneficial in small networks, of Trust-1 type, however in larger networks some paths can readily be discarded as they contribute little to the final confidence parameter.

The table shown below summarizes the performance runs of the software prototype:

Scenario	Run1	Run2	Run3	Run4	Run5	Avg.	Comment
Example4_3.xml	70ms	80ms	80ms	90ms	90ms	82ms	3 nodes, 1 certification path
Example4_4.xml	110ms	120ms	110ms	110ms	111ms	112.2ms	4 nodes, 2 certification paths
Example4_5.xml	131ms	110ms	140ms	130ms	130ms	128.2ms	5 nodes, 2 certification paths
Example4_6.xml	160ms	110ms	120ms	120ms	120ms	126ms	4 nodes, 3 certification paths
Example4_7.xml	120ms	140ms	130ms	131ms	140ms	132.2ms	4 nodes, 4 certification paths

It is important to note that the measurements show that the execution time is very short - on the order of 100ms. This is a length of time which can't be measured accurately. In non-real time operating systems, there are numerous factors influencing the execution time of programs. The execution time varied a lot from execution to execution, and it can be seen that the setup overhead time - memory allocation etc. - is comparable with the execution time of the algorithm. Execution time for 1 certification path was 82ms on average, and it was 112.2ms for two certification paths. The main conclusion that can be reliably made about the algorithm is that the execution time is very short, and that it scales reasonably well. Optimizations to the path search algorithm - sensitivity analysis can definitely optimize scalability further.

7.2 Contributions made to the field

This thesis demonstrates a complete design and implementation of a software prototype for evaluating the probabilistic confidence valuation concept. Peer to Peer, ad-hoc environments present a new challenge to designing an effective PKI. Probabilistic confidence valuation may prove to be the solution that enables numerous applications utilizing next generation wireless hand held devices.

A set of algorithms for realizing the core concept behind the Peer to Peer PKI is presented. The algorithms for handling the probabilistic confidence valuation syntax, finding certification and trust paths and calculating the confidence value are presented.

7.3 Further research

Further work on the prototype could be done in order to move closer to a practical implementation of a Peer to Peer PKI. A number of development paths are outlined below:

- elaborate XML data exchange protocol
- algorithm for handling higher level trust
- JXTA framework implementation
- Extend the path finding algorithm with sensitivity analysis, so that only the paths contributing most to the end result are considered

The prototype nodes use XML queries to ask for information from other nodes in the network. This could be extended to a more general XML based protocol, allowing for peer version numbering, capabilities, service advertisement etc.

The design phase identified a problem with the algorithm for finding higher level trust paths - an algorithm that would find a set of optimal trust paths in the network would require an exhaustive search to be performed. This is unacceptable in any practical implementation, therefore further research is needed to overcome this problem.

A P2P harness using JXTA's service advertisement and discovery could be used in a final implementation of a Peer to Peer PKI

Finally, although the prototype is quite fast with execution times on the order of 100ms-160ms, the scalability and speed could be improved further by introducing sensitivity analysis to the certification path search. The algorithm should be able to discard paths which contribute least to the final confidence parameter value.

Chapter 8

Summary

Implementing a PKI in a Peer to Peer environment presents a challenge. The wireless Peer to Peer environment lacks the infrastructure necessary to implement a X.509-style PKI. X.509 requires the presence of a hierarchy of certification servers - a root server with a self-signed certificate and regional servers whose certificates are signed by the root server. These servers then issue signed certificates for individual entities. In an ad-hoc environment we cannot assume that there will be a connection to the infrastructure certificate servers. Therefore there is a need to introduce a new concept - the Peer to Peer PKI.

The P2P PKI is based on a core concept introduced by Ueli Maurer - probabilistic confidence valuation. This idea builds on the PGP Web of Trust concept, where the usual certification hierarchy is replaced by a chain of certificates. Each entity issues their own self-signed certificate, and contributes to the system by signing other entities' certificates. An entity wishing to determine the validity of another entitie's certificate needs to find a complete chain of signatures leading to the other entity. Probabilistic confidence valuation extends this concept in two ways. First of all the concept of trust is introduced. In addition to finding a certification chain, the entity needs to trust each node along the way. The other enhancement is the definition of the probability that a given statement - be it authenticity or trust - is valid. Therefore each statement in the certification and trust chains has a probability assigned. This can then be used to determine the overall confidence parameter by solving the inclusion-exclusion equation.

This thesis presents the design, implementation and evaluation of a software prototype. The prototype is used to evaluate the core concept behind the Peer to Peer PKI - the probabilistic confidence valuation. In a given network of nodes, and given probabilities assigned to the individual statements a set of algorithms is developed to calculate the confidence parameter for the statement $Aut_{A,B}$ - Alice's belief that Bob's certificate is authentic.

The design of the software prototype pointed to some significant findings. The design and implementation of algorithms for finding the confidence parameter for a level 1 trust network is relatively easy. The algorithms for finding certification and trust paths are based on the well known and efficient Bellman-Ford algorithm. The inclusion-exclusion equation can be calculated using a recursive algorithm presented in this thesis. However, the introduction of higher level trust complicates the valuation of the confidence parameter. Higher level trust statements imply the existence of lower level trust statements, which introduces interdependencies in the trust graph of the network. An algorithm that would find optimum trust paths would requires exhaustive search time, which is acceptable theoretically, but not in a practical implementation.

Future research in this area could concentrate on improving scalability of the algorithm by introducing sensitivity analysis to the path finding algorithms. Additionally, theoretical work could be done in order to find a less-than-exhaustive algorithm for determining the trust path in a higher level trust network.

Bibliography

[1] Internet2 consortium http://www.internet2.edu/about/ Last visited on 11.07.2004

[2] SUNET Internet2 Land Speed Record: 69.073 Pbmps http://proj.sunet.se/LSR2/. Last visited on 11.07.2004.

[3] Philip Zimmerman's Web of Trust http://www.rubin.ch/pgp/weboftrust.en.html. Last visited on 11.07.2004.

[4] Personal Communications Service (PCS) http://wireless.fcc.gov/services/broadbandpcs/. Last visited on 11.07.2004.

[5] Global System for Mobile Communications http://www.gsmworld.com/technology/gsm.shtml. Last visited on 11.07.2004.

[6] General Packet Radio Service (GPRS) http://www.gsmworld.com/technology/gprs/index.shtml. Last visited on 11.07.2004.

[7] Enhanced Data Rates for Global Evolution (EDGE) http://www.gsmworld.com/technology/edge/index.shtml. Last visited on 11.07.2004.

[8] Universal Mobile Telecommunications System (UMTS) http://www.umts-forum.org/. Last visited on 11.07.2004.

[9] CDMA2000 - a registered trademark of Qualcomm http://www.cdmatech.com/solutions/cdma2000_3g_solutions.jsp?L2=cdma2000_3g_solutions. Last visited on 11.07.2004.

[10] Wireless Fidelity / Wireless LAN (WLAN) http://www.wi-fi.org/OpenSection/why_Wi-Fi.asp?TID=2. Last visited on 11.07.2004.

[11] Bluetooth https://www.bluetooth.org/spec/. Last visited on 11.07.2004.

[12] Home RF http://www.palowireless.com/homerf/about.asp Last visited on 11.07.2004

[13] IrDA http://www.irda.org/displaycommon.cfm?an=1. Last visited on 11.07.2004.

[14] Near Field Communication (NFC) http://www.semiconductors.philips.com/markets/identification/products/nfc/. Last visited on 11.07.2004.

[15] Radio Frequency Identification http://www.aimglobal.org/technologies/rfid/. Last visited on 11.07.2004.

[16] Secure Socket Layer (SSL) http://wp.netscape.com/security/techbriefs/ssl.html. Last visited on 11.07.2004.

[17] W.Diffie and M.E.Hellman, "New directions in cryptography," IEEE Trans. Inform. Theory, IT-22, 6, 1976, pp.644-654. Available at: http://citeseer.ist.psu.edu/diffie76new.html

[18] L. Kohnfelder, "Towards a Practical Public-Key Cryptosystem", Available at: http://theses.mit.edu/Dienst/UI/2.0/Composite/0018.mit.theses/1978-29/1

[19] Ueli Maurer. "Modelling a public-key infrastructure." In Computer Security - ESORICS'96. Springer-Verlag, 1996. Available at: http://citeseer.ist.psu.edu/maurer96modelling.html

[20] Andy Oram. "Peer-to-Peer: Harnessing the Power of Disruptive Technologies"(Chapter 1). March 2001. Available at: http://www.oreilly.com/catalog/peertopeer/chapter/ch01.html

[21] R. L. Rivest and B. Lampson, "SDSI - A Simple Distributed Security Infrastructure". April 1996.
[22] C. Ellison et al., "RFC 2693 - SPKI Certificate Theory", September 1999, Available at: ftp://ftp.isi.edu/in-notes/rfc2693.txt
[23] Philip Zimmerman. http://www.philzimmermann.com/. Last visited on 11.07.2004.
[24] PGP Corporation. http://www.pgp.com/. Last visited on 11.07.2004.
[25] Rivest, R. L., Shamir, A., Adleman, L. A. "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, Vol.21, Nr.2, 1978, S.120-126. Available at: http://citeseer.ist.psu.edu/rivest78method.html
[26] The Freenet Project http://freenet.sourceforge.net/. Last visited on 11.07.2004.
[27] BitTorrent http://bitconjurer.org/BitTorrent/. Last visited on 11.07.2004.
[28] Red vs Blue http://www.redvsblue.com/. Last visited on 11.07.2004.
[29] Knoppix ISO images http://www.knoppix.net/get.php Last visited on 11.07.2004
[30] Akamai http://www.akamai.com/index_flash.html. Last visited on 11.07.2004.
[31] Google http://www.google.com/. Last visited on 11.07.2004.
[32] Network News Transport Protocol (NNTP) http://www.ietf.org/rfc/rfc977.txt. Last visited on 11.07.2004.
[33] TErrestrial Trunked RAdio (TETRA) http://www.tetramou.com/. Last visited on 11.07.2004.
[34] Mathworld. Inclusion-Exclusion principle http://mathworld.wolfram.com/Inclusion-ExclusionPrinciple.html Last visited on 11.07.2004
[35] Bruce Schenier. "Applied Cryptography: Protocols, Algorithms and Source Code in C". John Wiley & Sons Inc. November 1995.

Appendix A

Source Code

A.1 Statement.java

```
package p2p2ki;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;

/**
 * @author Tomasz Cholewinski s020054
 */

public class Statement {
public int type;
public int from;
public int to;
public int level;
public double value;

public Statement(){}

public Statement(int typ, int fro, int t, int leve, double valu) throws Exception
{
if(typ!=1 && typ!=3) new Exception("Statement must be of type Trust or Rec");
if(valu<0 || valu>1) new Exception("Value must be between 0 and 1");
type=typ; from=fro; to=t; level=leve; value=valu;
}

public Statement(int typ, int fro, int t, double valu) throws Exception
{
if(typ!=0 && typ!=2) new Exception("Statement must be of type Aut or Cert");
if(valu<0 || valu>1) new Exception("Value must be between 0 and 1");
type=typ; from=fro; to=t; value=valu;
```

```
}

public boolean isEqual(Statement a)
{
if(a.type==type && a.from==from && a.to==to && a.level==level) return true;
else return false;
}

public boolean equals(Object anObject)
{
if((anObject.getClass()==Statement.class)&&(this.isEqual((Statement)anObject)))
return true;
else return false;
}

public boolean isParallel(Statement a)
{
if(a.from==from && a.to==to) return true;
else return false;
}

public void Print()
{
if(this==null) System.out.println("Undefined");
System.out.print(" ");
switch(this.type)
{
case 0: System.out.print("Aut"+this.value+" ");break;
case 1: System.out.print("Trust"+this.value+" ");break;
case 2: System.out.print("Cert"+this.value+" ");break;
case 3: System.out.print("Rec"+this.value+" ");break;
case -1: System.out.print("Stub ");break;
default: System.out.print("Unk ");break;
}
System.out.print(""+this.from+"->"+this.to);
}

public void Println()
{
Print();System.out.println();
}

public View getTrustPath(View VSa) throws Exception
{
try
{
BellmanFord BelF =
new BellmanFord(
```

```
new Network(VSa.getTrustRec()), VSa.getTrustRec(), this.from);
if(BelF.modifiedBF()) {return BelF.getView();}
else return new View();
}
catch(Exception e) {throw new Exception(e);}

}

public int hashCode()
{
String Return = ""+type+from+to+level+value;
return Return.hashCode();
}

public byte[] toByteArray() throws IOException
{
ByteArrayOutputStream result = new ByteArrayOutputStream();
result.write(type);
result.write(from);
result.write(to);
result.write(level);
result.write(String.valueOf(value).getBytes());
return result.toByteArray();
}

public Statement(byte[] bytes)
{
ByteArrayInputStream result = new ByteArrayInputStream(bytes);
type=result.read();
from=result.read();
to=result.read();
level=result.read();
byte value[] = new byte[result.available()];
result.read(value,0,result.available());
this.value= (new Double(new String(value))).doubleValue();

}

/**
 * Returns the from.
 */
public int getFrom() {
return from;
}
/**
 * The from to set.
 */
public void setFrom(int from) {
```

```
this.from = from;
}
/**
 * Returns the level.
 */
public int getLevel() {
return level;
}
/**
 * The level to set.
 */
public void setLevel(int level) {
this.level = level;
}
/**
 * Returns the to.
 */
public int getTo() {
return to;
}
/**
 * The to to set.
 */
public void setTo(int to) {
this.to = to;
}
/**
 * Returns the type.
 */
public int getType() {
return type;
}
/**
 * The type to set.
 */
public void setType(int type) {
this.type = type;
}
/**
 * Returns the Value.
 */
public double getValue() {
return value;
}
/**
 * The Value to set.
 */
public void setValue(double value) {
```

```
this.value = value;
}
}
```

A.2 View.java

```
package p2p2ki;

/**
 * @author Tomasz Cholewinski s020054
 */

public class View {

public Statement statements[];

public View(){}

public View(int length)
{
statements = new Statement[length];
}

public View(Statement a[])
{
statements = new Statement[a.length];
for(int i=0; i<a.length; i++) statements[i]=a[i];
}

public View(View a)
{
statements = new Statement[a.statements.length];
for(int i=0; i<a.statements.length; i++) statements[i]=a.statements[i];
}


public View Add(View Va)
{
if(this.statements==null || this.statements.length==0)
if(Va.statements==null || Va.statements.length==0) return new View();
else return Va;

if(Va.statements==null || Va.statements.length==0)
if(this.statements==null || this.statements.length==0) return new View();
else return this;

View V = new View(Va.statements.length+this.statements.length);
```

```
for(int i=0; i<Va.statements.length; i++) V.statements[i]=Va.statements[i];
for(int i=0; i<this.statements.length; i++)
V.statements[Va.statements.length+i]=this.statements[i];

//Worst case result length is the sum of the lengths of the two views.
View Result = new View(Va.statements.length+this.statements.length);

int position=0; boolean found=false;

Result.statements[position++]=V.statements[0];
for(int i=1; i<V.statements.length; i++)
for(int j=0; j<position; j++)
{
if(V.statements[i].isEqual(Result.statements[j])) j=position;
else if(j==position-1) Result.statements[position++]=V.statements[i];
}

//Truncate the result.
return Result.Truncate(position);
}



public View getAutCert()
{
if(this.statements.length==0) return new View();
View Set=this;
int position=0;
View Result = new View(Set.statements.length);
for(int i=0; i<Set.statements.length; i++)
{
if(Set.statements[i].type==0 || Set.statements[i].type==2)
Result.statements[position++]=Set.statements[i];
}

return Result.Truncate(position);
}

public int getMaxTrustLevel()
{
if(this.statements==null) return 0;
int max_trust=0;
for(int i=0; i<this.statements.length; i++)
if(this.statements[i].level>max_trust)
max_trust=this.statements[i].level;
return max_trust;
}
```

```
public double getMultiplied()
{
if(this!=null)
{


double Result=1;
for(int i=0; i<statements.length; i++)
if(this.statements[i].type==1||this.statements[i].type==3)
{
int max_trust=0;
for(int j=0; j<this.statements.length; j++)
if(this.statements[j].isParallel(this.statements[i]) && this.statements[j].level>max_trust)
max_trust=this.statements[j].level;
if(this.statements[i].level==max_trust) Result*=statements[i].value;

}
else Result*=statements[i].value;
return Result;
}
else return 0.0;

}

public View getTrustRec()
{
if(this.statements.length==0) return new View();
View Set=this;
int position=0;
View Result = new View(Set.statements.length);
for(int i=0; i<Set.statements.length; i++)
{
if(Set.statements[i].type==1 || Set.statements[i].type==3)
Result.statements[position++]=Set.statements[i];
}

return Result.Truncate(position);
}

public View getTrustView(View CertPath, int dest) throws Exception
{
try
{
View Result = new View();
View TrustNet = this.getValidTrustNet();

for(int i=0; i<CertPath.statements.length; i++)
if(CertPath.statements[i].from!=0 && CertPath.statements[i].from!=dest)
```

```
{
View Trust = CertPath.statements[i].getTrustPath(this);
if(Trust.statements!=null && Trust!=null) Result=Result.Add(Trust);
else return new View();
}


return Result.RemoveStubs();
}
catch(Exception e) {throw new Exception(e);}
}

public View getValidTrustNet()
{
if(this.statements==null) return new View();

//Prepare the result buffer
View Result = this.getTrustRec();
int position=0;

//Get the nmax maximum trust level in the view
int nmax=0;
for (int i=0; i<this.statements.length; i++)
if((this.statements[i].type==1 || this.statements[i].type==3)
&&this.statements[i].level>nmax) nmax=this.statements[i].level;

//If no trust statements were found in the view, return an empty view
if(nmax==0) return new View();

//This could use a network adjacency matrix...
//Iterate through the network nmax times
for(int i=0; i<nmax; i++)
{
for(int j=0; j<Result.statements.length; j++)
{
boolean found=false;
if(Result.statements[j].type==-1||Result.statements[j].type==1) found=true;
for(int k=0; k<Result.statements.length; k++)
{

if(Result.statements[k].to==Result.statements[j].from &&
Result.statements[k].level>Result.statements[j].level)
{found=true; break;}
}
//if no valid trust neighbours were found, insert a stub instead of the statement
if(!found) Result.statements[j]=new Statement();

}
```

```
}

return Result.RemoveStubs();
}

public boolean isEqual(View V)
{
if(this.statements.length!=V.statements.length) return false;
for(int i=0; i<V.statements.length; i++)
if(!(this.statements[i].isEqual(V.statements[i]))) return false;
return true;
}

public void Print()
{
if(this.statements==null) System.out.println("Null");
else
{
for(int i=0; i<this.statements.length; i++) this.statements[i].Print();
System.out.println();
System.out.println(" (Multiplied value is "+this.getMultiplied()+")");
}

}

public View Remove(Statement S)
{
if(this.statements==null) return new View();
if(S==null) return this;

if(this.statements.length>=1)
{
if(this.statements.length==1 && this.statements[0]==S) return new View();
if(this.statements.length==1 && this.statements[0]!=S) return this;

View Result = new View(this.statements.length-1);
int pos=0;
for(int i=0; i<this.statements.length; i++)
if(this.statements[i]!=S) Result.statements[pos++]=this.statements[i];

return Result;
}
//if the this View is empty return a new empty view.
return new View();
}

public View RemoveStubs()
{
```

```
if(this.statements!=null)
{
View Result = new View(this.statements.length);
int position=0;
for(int i=0; i<this.statements.length; i++)
if(this.statements[i]!=null)
Result.statements[position++]=this.statements[i];
return Result.Truncate(position);
}
else return new View();

}

public View Truncate(int position)
{
if(position==0) return new View();
View ActualResult = new View(position);
for(int i=0; i<position; i++) ActualResult.statements[i]=this.statements[i];
return ActualResult;
}

}
```

A.3 Network.java

```
package p2p2ki;

/**
 * @author Tomasz Cholewinski s020054
 */

public class Network {

public class Vertex
{
public Statement edge[];
public Vertex() {}
public Vertex(int a) {edge=new Statement[a];}
}
public int vertex_hi;

public Vertex vertices[];

public Network(int a)
{
vertices=new Vertex[a];
```

```
}

public Network(View V) throws Exception
{
try
{
if(V.statements==null) return;

/*
 * Scan for the highest nunmbered vertex in the View
 * The source vertex S always has the id of 0.
*/
vertex_hi=0;
for(int i=0; i<V.statements.length; i++)
{
if(V.statements[i].from>vertex_hi) vertex_hi=V.statements[i].from;
if(V.statements[i].to>vertex_hi) vertex_hi=V.statements[i].to;
}

vertices = new Vertex[vertex_hi+1];
for(int i=0; i<=vertex_hi; i++)
{
int type_tmp[]=new int[256];
int to_tmp[]=new int[256];
double value_tmp[]=new double[256];
int level_tmp[]=new int[256];
int position=0;

for(int j=0; j<V.statements.length; j++)
if(V.statements[j].from==i)
{
to_tmp[position++] = V.statements[j].to;
value_tmp[position-1] = V.statements[j].value;
level_tmp[position-1] = V.statements[j].level;
type_tmp[position-1] = V.statements[j].type;
}
vertices[i]= new Vertex();
if(position!=0)
{
vertices[i].edge = new Statement[position];
for(int k=0; k<position; k++)
vertices[i].edge[k]=new Statement(type_tmp[k], i, to_tmp[k], level_tmp[k], value_tmp[k]);
}
else
{
Statement s[] = {new Statement()};
vertices[i].edge = s;
}
```

```
}
}
catch(Exception e) {throw new Exception(e);}
}


public View getCertPath(Network N, View V, int dest)
{
//Bellman-Ford algorithm
BellmanFord BelF = new BellmanFord(N,V.getAutCert(),dest);

//Check if a shortest route can be found
if(BelF.BF()) return BelF.getView();
else return new View();
}

public void Print()
{
for(int i=0; i<this.vertices.length; i++)
{
System.out.print("Vertex "+ i + " :");
for(int j=0; j<this.vertices[i].edge.length; j++)
System.out.print(" "+this.vertices[i].edge[j].to+",");
System.out.println();
}
}
}
```

A.4 Paths.java

```
package p2p2ki;

/**
 * @author Tomasz Cholewinski s020054
 */

public class Paths {

private static View Result[] = new View[256];
private static int position=0;

/*
 * The purpose of the recursive getPaths function is to take an initial view,
 * determine if there is a shortest path from node 0 to node dest and iterate
 * recursively through the path removing statements - "breaking" the shortest path
 * in order to obtain the next best shortest path, then "breaking" it and so on. The
 * recursion proceeds to such a depth that a new shortest path cannot be found so that
```

```
 * eventually all unique paths are found.
 */

public static boolean getPaths(View V, int dest) throws Exception
{
try
{
Network N = new Network(V.getAutCert());
if(N.vertex_hi<dest) return false;
View Res = N.getCertPath(N, V, dest);
if(Res.statements==null) return false;
else
{
View Trust=V.getTrustView(Res,dest);
if (Trust.statements!=null && isUnique(Res.Add(Trust))) Result[position++]=Res.Add(Trust);
for (int i=0; i<Res.statements.length; i++)
getPaths(V.Remove(Res.statements[i]), dest);
return true;
}
}
catch(Exception e) {throw new Exception(e);}
}

public static void Init()
{
Result = new View[256];
position = 0;
}

public static void Print()
{
for(int i=0; i<position; i++)
Result[i].Print();
}

public static boolean isUnique(View V)
{
for(int z=0; z<position; z++)
{
if(Result[z].isEqual(V)) return false;
}
return true;
}

public static boolean getResults(View V, int dest) throws Exception
{
try
{
```

```
Init();
if(getPaths(V, dest))
{
View ActualResult[] = new View[position];
for(int i=0; i<position; i++)
{
ActualResult[i]=Result[i];
System.out.print("Path "+(i+1)+":");
ActualResult[i].Print();
}
double result=permutations.solveEquation(ActualResult);
System.out.println("P(Aut(0->"+dest+"))="+result);
return true;
}
else return false;
}
catch(Exception e) {throw new Exception(e);}
}
}
```

A.5 BellmanFord.java

```
package p2p2ki;

/**
 * @author Tomasz Cholewinski s020054
 */

public class BellmanFord {

private class MDBF { double dd[]; int pi[]; MDBF(){}}
public int d[];
public double dd[];

public int dest;
public Network N;
public int pi[];
public View V;

/*
 * The initializer for the BellmanFord class. Takes a View Vi,
 * a network Ne (obtained from the view Vi) and an integer destination dest.
 */
public BellmanFord(Network Ne, View Vi, int des)
{
if(Ne.vertices==null || Vi==null) return;
N = Ne;
```

```
V = Vi;
dest = des;
d = new int[N.vertices.length+1];
pi = new int[N.vertices.length+2];
dd = new double[N.vertices.length+1];
}

public boolean BF()
{
if(d==null) return false;
for(int i=0; i<=N.vertices.length; i++)
{
d[i]=65535;
pi[i]=-1;
}
d[0]=0;
pi[N.vertices.length+1]=dest;

for(int i=1; i<=N.vertices.length-1; i++)
{
for(int j=0; j<N.vertices.length; j++)
for(int k=0; k<N.vertices[j].edge.length; k++)
if(d[N.vertices[j].edge[k].to]>(d[j]+1))
{
d[N.vertices[j].edge[k].to]=d[j]+1;
pi[N.vertices[j].edge[k].to]=j;
}
}
if(N.vertex_hi<dest || d[dest]==65535) return false;
return true;
}

public View getView()
{
View Result = new View(V.statements.length);
int position=0;
int to_node=pi[N.vertices.length+1];
int from_node=pi[dest];
if(N.vertex_hi<dest || (d[dest]==65535 && dd[dest]==0.0)) return new View();
while(from_node!=-1)
{
for(int j=0; j<V.statements.length; j++)
if(V.statements[j].from==from_node && V.statements[j].to==to_node)
Result.statements[position++]=V.statements[j];

to_node=pi[to_node];
from_node=pi[from_node];
}
```

```
return Result.Truncate(position);
}

public boolean modifiedBF()
{
if(dd==null) return false;
for(int i=0; i<=N.vertices.length; i++)
{
dd[i]=0.0;
pi[i]=-1;
}
dd[0]=1.0;
pi[N.vertices.length+1]=dest;
for(int i=1; i<=N.vertices.length-1; i++)
{
for(int j=0; j<N.vertices.length; j++)
for(int k=0; k<N.vertices[j].edge.length; k++)
if(dd[N.vertices[j].edge[k].to]<(dd[j]*N.vertices[j].edge[k].value))
{
dd[N.vertices[j].edge[k].to]=dd[j]*N.vertices[j].edge[k].value;
pi[N.vertices[j].edge[k].to]=j;
}
}
if(N.vertex_hi<dest || dd[dest]==0.0) return false;
return true;
}

/*
 * Functions Print() and Printdd() are used to print out
 * the contents of the instance of the BellmanFord object.
 * Used for debugging and verbose output.
 */

public void Print()
{
N.Print();
V.Print();
System.out.print("d: ");
for(int i=0; i<d.length; i++) System.out.print(" "+d[i]);
System.out.println();
System.out.print("pi: ");
for(int i=0; i<pi.length; i++) System.out.print(" "+pi[i]);
}

public void Printdd()
{
N.Print();
V.Print();
```

```
System.out.print("dd: ");
for(int i=0; i<dd.length; i++) System.out.print(" "+dd[i]);
System.out.println();
System.out.print("pi: ");
for(int i=0; i<pi.length; i++) System.out.print(" "+pi[i]);
}

}
```

A.6 Node.java

A.7 PNode.java

```
package p2p2ki;
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.StringReader;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.security.interfaces.RSAPublicKey;
import java.util.Enumeration;
import java.util.Hashtable;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;



/**
 * @author Tomasz Cholewinski s020054
 */

public class PNode implements Runnable{
```

```
public int listenPort;
public int connectPorts[];
public String connectAddresses[];
public int NodeName;
public int SeekNode;
public String certificates[];
public boolean listening;
public Hashtable AutStore = new Hashtable();
public Hashtable extStatementStore = new Hashtable();
public Hashtable extAutStore = new Hashtable();
public Hashtable statementStore = new Hashtable();
public PrivateKey priv_key;
public View Sa = new View();

private DocumentBuilder parser;


public PNode(){}

public void setConnectPorts(int ports[])
{
if(ports!=null)
{
connectPorts = new int[ports.length];
for(int i=0; i<ports.length; i++) connectPorts[i]=ports[i];
listening=false;
}
}

public void setConnectAddresses(String addresses[])
{
if(addresses!=null)
{
connectAddresses = new String[addresses.length];
for(int i=0; i<addresses.length; i++) connectAddresses[i]=addresses[i];
listening=false;
}
}


public int[] getConnectPorts()
{
if(!listening) return connectPorts;
else return null;

}

public void setNodeName(int Name){NodeName=Name;}
```

```
public void setSeekNode(int Name){SeekNode=Name;}

public void setListenPort(int port)
{
if(port!=0)
{
listenPort=port;
listening=true;
}
}

public int getListenPort()
{
if(listening) return listenPort;
else return 0;
}

public boolean Process(String is)
throws SAXException, IOException, ParserConfigurationException
{
        // Create a DOM builder and parse the XML fragment
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
Document d = factory.newDocumentBuilder().parse(new InputSource(new StringReader(is)));

NodeList nodes = d.getDocumentElement().getChildNodes();
Node node_i = nodes.item(0);
if (node_i.getNodeType() == Node.ELEMENT_NODE
&& ((Element) node_i).getTagName().equals("DestNode")
&& (((Element) node_i).getAttributes().item(0).getNodeValue()!=String.valueOf(NodeName)))
return true;
return false;
}


public Socket getServerSocket(int port) throws IOException
{
ServerSocket srv = new ServerSocket(port);
Socket socket = srv.accept();
return socket;
}

public void run()
{
try
{
if(listening)
{
```

```
Runnable runnable = new PNodeServer(NodeName,listenPort,this);
Thread ListenServer = new Thread(runnable);
ListenServer.start();
ListenServer.join(5000); //4 second timeout
if(ListenServer.isAlive())
{
System.out.println(NodeName+
": Timeout occurred; Server "+NodeName+"-0 has not finished. Stopping.");
            ListenServer.stop();
}
else System.out.println(NodeName+": Server "+NodeName+"-0 has finished");
}
else
{
Thread ConnectClient[] = new Thread[connectAddresses.length];
for(int i=0; i<connectAddresses.length; i++)
{
Runnable runnable =
new PNodeClient(NodeName, i+1, connectAddresses[i], connectPorts[i], SeekNode, this);
ConnectClient[i] = new Thread(runnable);
ConnectClient[i].start();


}
for(int i=0; i<connectAddresses.length; i++)
{
long delayMillis = 5000; // 5 second timeout

ConnectClient[i].join(delayMillis);
        if (ConnectClient[i].isAlive())
        {
            System.out.println(NodeName+
             ": Timeout occurred; Client "+NodeName+"_"+i+" has not finished. Stopping.");
            ConnectClient[i].stop();
        }
        else
        {
         System.out.println(NodeName+": Client "+NodeName+"-"+i+" has finished");
        }
}
/*
 * Once all the client threads have succesfully or unsuccesfully exit, collate
 * the results into a usable view.
 */
View Sa = new View();
for (Enumeration e = extStatementStore.elements() ; e.hasMoreElements() ;)
         {
         Statement statement[]={(Statement)e.nextElement()};
```

```
         Sa=Sa.Add(new View(statement));
         }
for (Enumeration e = statementStore.elements() ; e.hasMoreElements() ;)
         {

         Statement statement[]={(Statement)e.nextElement()};
         Sa=Sa.Add(new View(statement));
         }
this.Sa=Sa;
}

}
catch (InterruptedException e) {System.out.println(e);}
catch (Exception e) {System.out.println(e);}
}

// Used to read a certificate
public void setCertificates(String certnames[])
{
try
{
certificates = new String[certnames.length];
for(int i=0; i<certnames.length; i++)
{
certificates[i]=certnames[i];
RSAPublicKey publicKey; //Used to store the public key in the object
X509Certificate Cert; //used to store the X509 certificate object
String Name; //Contains the name read from the public certificate

FileInputStream fis = new FileInputStream(certnames[i]);
BufferedInputStream bis = new BufferedInputStream(fis);
CertificateFactory cf = CertificateFactory.getInstance("X.509","BC");
Cert = (X509Certificate)cf.generateCertificate(bis);
publicKey = (RSAPublicKey)Cert.getPublicKey();
String CertS = Cert.getSubjectDN().toString();
//Acquire the node name stored in the certificate
Name = CertS.split("CN=")[1].split("[,\n\r\f]")[0];
//Store the new certificate in the AutStore hashtable
AutStore.put(Name, publicKey);
}
}
catch (Exception e) {System.out.println(e);e.printStackTrace();}
}

synchronized public void addCertificate(String Name, RSAPublicKey publicKey)
{
extAutStore.put(Name, publicKey);
}
```

```
synchronized public void addExtStatement(Statement statement)
{
extStatementStore.put(String.valueOf(statement.hashCode()),statement);

}

/**
 * The statementStore to set.
 */
public void setStatementStore(Statement statements[])
{

for(int i=0; i<statements.length; i++)
statementStore.put(String.valueOf(statements[i].hashCode()),statements[i]);
}

public String[] getCertificates()
{
return certificates;
}

public PublicKey getCertificate(String Name)
{
return (PublicKey)AutStore.get(Name);
}

public int getName()
{
return NodeName;
}

/**
 * Returns the connectAddresses.
 */
public String[] getConnectAddresses()
{
return connectAddresses;
}
/**
 * Returns the statementStore.
 */
public Statement[] getStatementStore()
{
Statement statement[] = new Statement[statementStore.size()]; int position=0;
for (Enumeration e = statementStore.elements() ; e.hasMoreElements() ;)
statement[position++]=(Statement)e.nextElement();
return statement;
```

```
}
/**
 * Returns the nodeName.
 */
public int getNodeName() {
return NodeName;
}
/**
 * Returns the seekNode.
 */
public int getSeekNode() {
return SeekNode;
}
}
```

A.8 PNodeClient.java

```
package p2p2ki;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.math.BigInteger;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.SocketAddress;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.RSAPublicKeySpec;
import java.util.StringTokenizer;

/**
 * @author Tomasz Cholewinski s020054
 */

public class PNodeClient implements Runnable{
public String connectAddress="localhost";
public int connectPort=8667;
public int NodeName;
public int SeekNode;
```

```
public int ConnectNodeName;
private PNode hook;

public PNodeClient(){}
public PNodeClient(int NodeName, int ConnectNodeName,
String connectAddress, int connectPort, int SeekNode, PNode hook)
{
this.NodeName=NodeName;
this.connectPort=connectPort;
this.connectAddress=connectAddress;
this.SeekNode=SeekNode;
this.hook=hook;
this.ConnectNodeName=ConnectNodeName;
}

public void run()
{
try
{
InetAddress ConnectAddress = InetAddress.getByName(connectAddress);
    SocketAddress sockaddr = new InetSocketAddress(ConnectAddress, connectPort);
    Socket tx_socket = new Socket();
    tx_socket.setReceiveBufferSize(256);
    int timeoutMs = 2000;   // 2 second timeout
    tx_socket.connect(sockaddr, timeoutMs);
    BufferedWriter wr = new BufferedWriter(new OutputStreamWriter(tx_socket.getOutputStream()));
    BufferedReader rd = new BufferedReader(new InputStreamReader(tx_socket.getInputStream()));
    wr.write("<Query><DestNode Name=\""+SeekNode+"\"/></Query>\n");
    wr.flush();
    String str;

    while ((str = rd.readLine()) != null)
            {
     if(str.startsWith("PublicKey"))
{
     RSAPublicKeySpec KeySpec =
     new RSAPublicKeySpec(
     new BigInteger(getBytes(str.split(":")[1].split("=")[2])),
new BigInteger(getBytes(str.split(":")[1].split("=")[3])));
KeyFactory kf = KeyFactory.getInstance("RSA","BC");
RSAPublicKey publicKey = (RSAPublicKey)kf.generatePublic(KeySpec);
hook.addCertificate(str.split(":")[1].split("=")[1],publicKey);
             System.out.flush();
     }
     else if(str.startsWith("Statement"))
     {
     Statement statement = new Statement(getBytes(str.split(":")[1]));
     hook.addExtStatement(statement);
```

```
     }

            }
    tx_socket.close();
}
catch (IOException e) {System.out.println(NodeName+": "+e);System.out.flush();}
catch (NoSuchAlgorithmException e) {System.out.println(NodeName+": "+e);System.out.flush();}
catch (NoSuchProviderException e) {System.out.println(NodeName+": "+e);System.out.flush();}
catch (InvalidKeySpecException e) {System.out.println(NodeName+": "+e);System.out.flush();}
}

private static byte[] getBytes( String str )
{
ByteArrayOutputStream bos = new ByteArrayOutputStream();
StringTokenizer st = new StringTokenizer( str, "-", false );
while( st.hasMoreTokens() )
{
int i = Integer.parseInt( st.nextToken() );
bos.write( ( byte )i );

}
return bos.toByteArray();
}

}
```

A.9 PNodeServer.java

```
package p2p2ki;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.interfaces.RSAPublicKey;
import java.util.Enumeration;

/**
 * @author Tomasz Cholewinski s020054
 */

public class PNodeServer implements Runnable{

public int ListenPort=8670;
public int NodeName;
```

```
public PNode hook;

public PNodeServer(){}
public PNodeServer(int NodeName, int ListenPort, PNode hook)
{
this.NodeName=NodeName;
this.ListenPort=ListenPort;
this.hook=hook;
}


public void run()
{
try
{

ServerSocket srv = new ServerSocket(ListenPort);
        Socket rcv_socket = srv.accept();
        BufferedWriter wr =
         new BufferedWriter(new OutputStreamWriter(rcv_socket.getOutputStream()));
        BufferedReader rd =
         new BufferedReader(new InputStreamReader(rcv_socket.getInputStream()));
        String str= rd.readLine();
    if(hook.Process(str))
        {

         for (Enumeration e = hook.AutStore.elements() ; e.hasMoreElements() ;)
         {
         RSAPublicKey publicKey=(RSAPublicKey)e.nextElement();
         String Name=(String)hook.AutStore.get(publicKey);
         wr.write("PublicKey:"+"="+Name+"="+
         getString(publicKey.getModulus().toByteArray())+
"="+getString(publicKey.getPublicExponent().toByteArray())+"\n");
         wr.flush();
            }

         for (Enumeration e = hook.statementStore.elements() ; e.hasMoreElements() ;)
         {
         Statement statement = (Statement)e.nextElement();
         wr.write("Statement:"+getString(statement.toByteArray())+"\n");
         wr.flush();
            }


        }
        rcv_socket.close();

}
```

```
catch(Exception e)
{System.out.println(NodeName+": "+e);System.out.flush();}

}

private static String getString( byte[] bytes )
{
StringBuffer sb = new StringBuffer();
if (bytes == null) return null;
for( int i=0; i<bytes.length; i++ )
{
byte b = bytes[ i ];
sb.append( ( int )( 0x00FF & b ) );
//Make sure that the int is no more than 8 significant bits
if( i+1 <bytes.length )
{
sb.append( "-" );
}
}
return sb.toString();
}




}
```

A.10 threaded_test_harness.java

```
package p2p2ki;

import java.beans.XMLDecoder;
import java.io.BufferedInputStream;
import java.io.FileInputStream;

/**
 * @author Tomasz Cholewinski s020054
 *
 * The threaded_test_harness.java file is the entry point for the implementation
 * of the trust propagation model for the Master's thesis by Tomasz Cholewinski.
 *
 * 2004
 *
 */

public class threaded_test_harness {
public static void main(String[] args) {
```

```
try
{
if(args.length!=1)
throw new Exception("Invalid arguments passed, please supply the scenario file name");

//Load the scenario file
        XMLDecoder decoder =
         new XMLDecoder(new BufferedInputStream(new FileInputStream(args[0])));
        PNode Nodes[] = (PNode[])decoder.readObject();
        decoder.close();

        Thread Threads[] = new Thread[Nodes.length];

        //This is the point where the start of the performance measurement is taken
        long StartTime = System.currentTimeMillis();
        //Initialize the Nodes (Instantiate the threads and run them)
for(int i=0; i<Nodes.length; i++)
{
System.out.println("Main: Running node "+Nodes[i].NodeName); System.out.flush();

Runnable runnable = Nodes[i];
Threads[i] = new Thread(runnable);
Threads[i].start();
}

//Wait for all the threads to finish executing, terminate them if timeout is hit
for(int i=0; i<Nodes.length; i++)
{
long delayMillis = 6000; // 6 second timeout
    try
    {
        Threads[i].join(delayMillis);
        if (Threads[i].isAlive())
        {
            System.out.println(
             "Main: Timeout occurred; thread "+Nodes[i].NodeName
+" has not finished. Stopping.");
            Threads[i].stop();
        }
        else
        {
         System.out.println("Main: Thread "+i+" has finished");
        }
    }
    catch (InterruptedException e) {e.printStackTrace();}
}
/*
 * Once all the threads have finished, there should reside a usable Sa view on Nodes[0].
```

```
 * This View can then be used to compute the trust propagation in the network
 */
System.out.println("\n====="+args[0]+"=====");
System.out.println("The view obtained from the network:");
Nodes[0].Sa.Print();
System.out.println("========================");

/*The static Paths.getResults performs the computation
*of the trust paths and trust propagation
*/
if(!Paths.getResults(Nodes[0].Sa, Nodes[0].getSeekNode()))
System.out.println("The given view did not contain a trust path");

//End of performance measurement
long RunTime = System.currentTimeMillis()-StartTime;
    System.out.println("Main: Total execution time was "+RunTime+" miliseconds");
    System.out.flush();
}
catch (Exception e) {e.printStackTrace();}
}
}
```

Appendix B

Scenario File

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.4.2_03" class="java.beans.XMLDecoder">
 <array class="p2p2ki.PNode" length="3">
  <void index="0">
   <object class="p2p2ki.PNode">
    <void property="certificates">
     <array class="java.lang.String" length="1">
      <void index="0">
       <string>1_certificate.pem</string>
      </void>
     </array>
    </void>
    <void property="connectAddresses">
     <array class="java.lang.String" length="2">
      <void index="0">
       <string>localhost</string>
      </void>
      <void index="1">
       <string>localhost</string>
      </void>
     </array>
    </void>
    <void property="connectPorts">
     <array class="int" length="3">
      <void index="0">
       <int>8667</int>
      </void>
      <void index="1">
       <int>8668</int>
      </void>
      <void index="2">
       <int>8669</int>
      </void>
     </array>
```

```
   </void>
   <void property="seekNode">
    <int>2</int>
   </void>
   <void property="statementStore">
    <array class="p2p2ki.Statement" length="2">
     <void index="0">
      <object class="p2p2ki.Statement">
       <void property="to">
        <int>1</int>
       </void>
       <void property="value">
        <double>0.9</double>
       </void>
      </object>
     </void>
     <void index="1">
      <object class="p2p2ki.Statement">
       <void property="level">
        <int>1</int>
       </void>
       <void property="to">
        <int>1</int>
       </void>
       <void property="type">
        <int>1</int>
       </void>
       <void property="value">
        <double>0.7</double>
       </void>
      </object>
     </void>
    </array>
   </void>
  </object>
 </void>
 <void index="1">
  <object class="p2p2ki.PNode">
   <void property="certificates">
    <array class="java.lang.String" length="1">
     <void index="0">
      <string>2_certificate.pem</string>
     </void>
    </array>
   </void>
   <void property="listenPort">
    <int>8667</int>
   </void>
```

```
    <void property="nodeName">
     <int>1</int>
    </void>
    <void property="statementStore">
     <array class="p2p2ki.Statement" length="1">
      <void index="0">
       <object class="p2p2ki.Statement">
        <void property="from">
         <int>1</int>
        </void>
        <void property="to">
         <int>2</int>
        </void>
        <void property="type">
         <int>2</int>
        </void>
        <void property="value">
         <double>0.8</double>
        </void>
       </object>
      </void>
     </array>
    </void>
   </object>
  </void>
  <void index="2">
   <object class="p2p2ki.PNode">
    <void property="listenPort">
     <int>8668</int>
    </void>
    <void property="nodeName">
     <int>2</int>
    </void>
    <void id="StatementArray0" property="statementStore"/>
    <void property="statementStore">
     <object idref="StatementArray0"/>
    </void>
   </object>
  </void>
 </array>
</java>
```