HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Telecommunications Software and Multimedia Laboratory

Anonymous Authorization in Networked Systems: An Implementation of Physical Access Control System

Master's Thesis

Pekka Kanerva

Helsinki University of Technology
Department of Computer Science
and Engineering
Telecommunications Software
and Multimedia Laboratory
Espoo 2001

Teknillinen korkeakoulu
Tietotekniikan osasto

Tietoliikenneohjelmistojen
ja multimedian laboratorio

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF MASTER'S THESIS

Author:	Pekka Kanerva	
Name of thesis:	Anonymous Authorization in Networked Systems: An Implementation of Physical Access Control System	
Date:	March 27, 2001	**Pages:** 12 + 74
Department:	Department of Computer Science and Engineering	**Chair:** Tik-110 (Computer networks)
Supervisor:	Professor Arto Karila	
Advisor:	Sanna Liimatainen, M.Sc.	

Fast network connections have made it possible to develop truly distributed applications. The Finnish university network FUNET's backbone has recently upgraded to 2.5 GB making it one of the world's fastest research networks. These fast networks have shortened the conceptual distances considerably which in turn has hastened the globalization of the society and the new society has brought along new problems.

In small communities, the members of the community were identified based on their names. The people knew each other beforehand and based their trust on each other on that knowledge. A centralized computer system is a bit similar to a small community in a sense that the identity of the person is known at least on some level. However, in a network, which crosses national borders, names no longer provide sufficient information on which trust could be based. Therefore, mere names should not be used as a basis of trust decisions in widely distributed systems.

Furthermore, the identity of the user is usually not the essential property which we want to know. Instead, the principal question is the authority of the user. Various research projects have supported this approach. A more expressive and extensive and yet secure way to describe authorization information is needed.

One such method for expressing trust relations and authorization information are SPKI certificates. Certificates bind authorization information directly to a public encryption key instead of first identifying the person and then checking the authorization. Recent studies have shown that an encryption key is a far more reasonable piece of information to be used as the basis of recognizing information than a person's name.

In this thesis I present a study of the problems concerning physical access control. In a large setting, a central administration of access control is impossible because it can scale up only to certain number of users. The distribution of the management is therefore required. Further, different parts of the organization might have different requirements for access control. The new opportunities and features of a distributed access control system are compared to the centralized solution. The benefits of distributing both the security policy management and access control permission evaluation are discussed and pointed out. The new system is even better than the old one and actually offers new advantages. The privacy of the users is maintained and the benefits of the distribution of the management tasks gives a more expressive tools for maintaining organization security.

Keywords:	authorization, access control, distributed system, certificate, SPKI, privacy protection
Language:	English

Tekijä:	Pekka Kanerva		
Työn nimi:	Anonyymi valtuutus verkotetuissa järjestelmissä: Fyysisen pääsynvalvontajärjestelmän toteutus		
Päivämäärä:	27.3.2001	**Sivuja:**	12 + 74
Osasto:	Tietotekniikan osasto	**Professuuri:**	Tik-110 (Tietokoneverkot)
Työn valvoja:	Professori Arto Karila		
Työn ohjaaja:	DI Sanna Liimatainen		

Nopeat tietoverkot ovat mahdollistaneet hajautettujen sovellusten kehittämisen. Suomen yliopistoverkko FUNETn runkoverkko on juuri päivitetty mahdollistamaan 2,5 GB siirtonopeuden joka samalla tekee siitä yhden maailman nopeimmista tutkimuskäytössä olevista verkoista. Nopeat maailmanlaajuiset verkkoyhteydet ovat samalla lyhentäneet etäisyyksiä ja nopeuttaneet yhteiskunnan globalisaatiota. Tämä kehitys on kuitenkin tuonut mukanaan uusia ongelmia.

Ihmiset elivät aikaisemmin pienissä yhteisöissä, joissa miltei kaikki tunsivat toisensa nimeltä. Ajan myötä ihmisten välille oli syntynyt erilaisia luottamussuhteita. Keskitetty tietojärjestelmä on jossain määrin verrattavissa tällaiseen yhteisöön. Keskitetyssä järjestelmässä käyttäjät voidaan tunnistaa heidän nimensä tai käyttäjätunnuksensa perusteella. Tämä ei päde enää valtakunnan rajat ylittävien tietoverkkojen suhteen. Nimi tai käyttäjätunnus ei enää takaa riittävää informaatiota luottamussuhteiden pohjaksi.

Käyttäjän tunnistaminen ei yleensä ole tavoiteltu asia. Sen sijaan tärkeämmässä asemassa ovat käyttäjän oikeudet ja valtuudet. Viimeisimmät tutkimustulokset osoittavat että tällainen lähestymistapa on parempi kuin identiteettiin perustuva. Käyttäjien valtuuksia on pystyttävä ilmaisemaan joustavammin ja hienosyisemmin kuin mihin käyttäjätunnuksia soveltamalla pystyttäisiin.

Eräs tällainen menetelmä ovat SPKI sertifikaatit, jotka sitovat valtuudet suoraan julkiseen salakirjoitusavaimeen identiteetin sijasta. Julkinen salakirjoitusavain on nimeä mielekkäämpi tapa esiintyä tietoverkoissa.

Olen tutkinut diplomityössäni fyysiseen pääsynvalvontaan liittyviä ongelmia. Keskitetty pääsynvalvonta ei ole riittävän joustava ratkaisu, mikäli organisaation koko kasvaa tietyn rajan yli. Tämän vuoksi myös pääsynvalvonnan hallinnointi on hajautettava. Organisaation eri osilla saattaa lisäksi olla erilaisia, toisistaan poikkeavia tarpeita ja vaatimuksia pääsynvalvonnan ja soveltamansa tietoturvapolitiikan suhteen. Olen vertaillut hajautetun pääsynvalvonnan mukanaan tuomia uusia ominaisuuksia keskitettyyn ratkaisuun, ja nostanut esiin hajautuksen mukanaan tuomia etuja. Hajautettu toteutus osoittautuu yhtä hyväksi kuin vanha, keskitetty ratkaisu ja hajautuksen myötä saavutetaan merkittäviä etuja. Käyttäjien yksityisyyden suoja säilyy ja hajautuksen myötä organisaation eri osien turvallisuus pääsynvalvonnan osalta voidaan taata paljon paremmin kuin aiemmin.

Avainsanat:	valtuutus, pääsynvalvonta, hajautettu järjestelmä, sertifikaatti, SPKI, yksityisyyden suoja
Kieli:	englanti

Contents

Acknowledgements

During my studies in Helsinki University of Technology, the awareness and general knowledge of information and computer security has increased a lot. Perhaps we are finally about to see the information security as an independent major subject.

This thesis was written as a part of the Telecommunications Software Security Architecture (TeSSA) research project in Telecommunications Software and Multimedia Laboratory. I want to thank the laboratory and my co-workers in the project for the opportunity to write my thesis during the project. Especial thanks to Yki Kortesniemi for the numerous conversations we had as well as suggestions for improvements and proofreading my text.

Especial thanks to my advisor Sanna Liimatainen who helped me to keep my thesis in direction and gave valuable advices and answered my numerous questions. Without her help my thesis would be of much lesser quality.

I also thank Professor Arto Karila, my supervisor, for the help and support he gave me during this thesis. Also, I appreciate the several long conversations which we had during my studies. From them I learned at least as much as I learned in his courses.

I thank Kristiina Volmari-Mäkinen for the reading samples of my text and giving advices to brush up my language.

I want to thank my parents and my sister for the encouragement and support they gave me during my studies. Without them it would have been impossible to take this work into an end.

Finally, I want to thank my friends for the encouragement they provided during this work. Especially I want to mention Tommi Elo, Juho Heikkilä, Anssi Kanninen, Jaakko Lehto, and Kaisu Mononen. Your support was invaluable in the hard times.

In Espoo, on April 9, 2001

Pekka Kanerva

Glossary

$\mathcal{P}(A)$	The power set of A, i.e. the set of all subsets of A.
c	An encrypted message $c = e(k, m)$. Also called a cipher text.
$e(k, m)$	An encryption transformation using the key k to encrypt the and message m. The corresponding decryption transformation is $d(k, c)$.
e_b^a	A user entity e_b which belongs to the security domain of Certification Authority CA$_a$.
h	A one-way hash function.
$h(k, m)$	A MAC calculated using a secret key k for a message m.
$h(m)$	A hash value of a message m.
k	A universal encryption or decryption key which is not specified in more detail.
k^+_{Alice}	Alice's public key in a public key cryptosystem.
k^-_{Alice}	Alice's private key in a public key cryptosystem.
m	A message which is transferred over an untrusted network.
AC	Access Control, also Attribute Certificate
ACL	Access Control List
ACM	Access Control Manager. The person who defines and administers the access control in an organizational unit.
ACMS	Access Control Management System. The whole system which includes the technical solutions, security policies, administrators, and end-users.
attack	An action taken by a malicious intruder that involves an exploitation of a vulnerability (or several vulnerabilities) in order to cause a threat to occur.

CA	Certification Authority, a network node responsible for establishing and giving statement about the authenticity of public encryption keys.
Cartesian product	Given two sets A and B, the Cartesian product is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$. Also called a cross product.
Cryptosystem	A set of cryptographic primitives, plaintext, ciphertext, keyspace, encrypting and decrypting transformations which as a whole provide security services.
DAC	Discretionary Access Control. A set of access control mechanisms which end-user herself can set or define.
DN	Distinguished Name. A name used in X.509 certificates which is assumed to be unique for every individual people.
DNS	Domain Name System
DSS	Digital Signature Scheme
GUI	Graphical User Interface
IEC	ISO/International Electrotechnical Commission
ISO	International Standards Organization
ITU-T	International Telecommunication Union – Telecommunication Standardization Sector
MAC	1) Mandatory Access Control. A system-wide set of access control mechanisms which the end-user cannot affect. 2) Message Authentication Code.
Party	A network node participating in a transaction by carrying out some protocol.
PDA	Personal Digital Assistant
PKC	Public Key Certificate. In context with X.509, the standard and RFCs define public key certificate to be an object which binds a public key to a distinguished name.
PKI	Public-Key Infrastructure, a certificate system, where you can have a certain amount of assurance that you have a valid public key of some principal.
RFC	Request For Comments. A set of de-facto standards concerning Internet and Internet protocols.

SD	Security Domain, a system controlled by one single authority to which every node in the system trust.
SHA	Secure Hash Algorithm
SSL	Secure Socket Layer. A public key cryptography method used widely in WWW-browsers.
TCB	Trusted Computing Base. The hardware and software mechanisms which enforce the security policy of a given system.
TeSSA	Telecommunication Software Security Architecture, an architecture which has been defined and studied in HUT.
threat	Any potential occurrence, malicious or not, that can have an undesirable effect on a computer system.
TTP	Trusted Third Party, a party of a protocol trusted by all of the protocol parties, which can be used as a verifier of correct protocol execution.
UML	Unified Modeling Language
vulnerability	An unfortunate characteristics in a computer system which makes it possible for a threat to potentially occur.

List of Figures

Chapter 1

Introduction

After the so called cold war between the super powers ended, spying and national espionage has nowadays concentrated on stealing business secrets and corporate information instead of military secrets. Information and knowledge is becoming the most valuable asset in business making. Therefore, companies have much more interest in controlling who can enter to the company facilities.

The very first layer of the overall security of an organization is the organizational and administrative security [79]. Physical access control to the facility is the key method to ensure that only authorized personnel can walk in to the company.

1.1 Motivation

Today it is becoming more customary that every action a person takes in a digital environment is linked with the identity of the person. Most of the access control decisions are based on the identity of the person. The development which have lead us to the current situation may have started in the history of mainframe computers. To access these mainframes, one needed a username and a password to authorize oneself to use the mainframe.

In a centralized system, identification based access control was adequate. However, in a distributed system, the plain identity tells very little of the access permissions the person has. A more general concept known as *trust management* is needed to express the various access permissions a person might have in a networked system.

While increasingly more activities of the society can take place remotely by using Internet, the general public is highly unaware of the clear traces they leave behind in the network. However, this applies also to the physical world. Credit cards, magnetic keys, all of these, when they are implemented electronically, are based on identity as the recognizing information. The city of Espoo has recently launched a project of 'member-of-the-city-card' which allows various services like paying the bus trips with the card. Based on identity, this new card makes it possible to collect information when and where individual people usually travel. Still, the basic task is to pay the trip, not to leave traces where we are traveling. Personal privacy is widely acknowledged as an intrinsic value [55, 73] and should be carefully protected. Technology should make our everyday living easier, not to create an Orwellian society as a side effect.

In this thesis, we have studied the possibility to leave the identities aside and express the physical access permissions in the form of *certificates* which bind the authorization directly to public key instead of any name. Our goal is to show that physical access control can be implemented safely without unnecessary identification as part of the procedure.

1.2 Organization of This Thesis

The rest of this thesis is organized as follows. Chapter 2 introduces the traditional building blocks of security in distributed systems. Chapter 3 discusses the trust as a phenomenon in human relations. It describes two common trust models used to secure communications over untrusted networks. Finally, it presents the recent results of trust management concept. In Chapter 4 various forms of digital certificates are studied and compared to each other. The last part of the Chapter introduces the cryptological methods which are applied in certificates. Chapter 5 studies the problems of access control on a general level and then goes into physical access control in more detail. Traditional, centralized version of access control is described and analyzed. A distributed solution is introduced and compared against the centralized one. Chapter 6 describes our proposed solution to the problem. Different subproblems of physical access control are described. Finally, an example of an access control decision is walked through as a demonstration of how our solution works in practice. Chapter 7 discusses the experiences of implementing the access control system and evaluates the tools which were used in the development process.

1.3 Notations Used

We use the following notations and naming conventions in this thesis. When we discuss cryptographic protocols, parties of such protocols, and cryptographic application programs we name the parties and users using English first names like “Alice”, “Bob”, “Carol”, etc.

File names and class names are written with fixed width font, for example: `/etc/passwd`. Class names are written in capital initials, for example: `MyClass`.

Certificate tags which denote permission information are written in Sans Serif font, for example: my-permission-tag.

Chapter 2

Distributed System Security

A distributed system is a group of computers that are connected to each other with some kind of network. These computers are usually called *nodes* and they communicate with each other through the network. However, nothing is presumed about the reliability of the network. In other words, we cannot rely that the message we sent to some node will be received by that node. This means that the transport medium, i.e. the physical network installation - routers, switches, cables, and so on - is *unreliable*. Even if the destination node receives our message, we still cannot be sure whether the message was tampered or whether the message was altered while the message was being routed to the receiving node. In this case, the network connection is said to be *untrusted*.

These problems raise the need for *distributed system security*, i.e. a number of different security services which guarantee that our message will be securely transmitted to its destination. There are a number of different ways to divide the security services into different groups. Amoroso defines three different sets of threats to the distributed system [1]. These threats are *disclosure* (compromising confidentiality), *integrity* (compromising content) and *denial of service* (compromising availability). Disclosure of information causes an unauthorized party to gain knowledge of the information being transferred. Integrity means that the information remains unaltered during transfer through network. An attack which aims to a denial of service threat to occur, results a situation in which the requested information is not available. In such a situation, even the authorized nodes cannot get the requested information.

These three security aspects are usually understood to be the three basic dimensions of computer security [83]. Any two of these cannot be used to express the third one. However, there are a few more security functions that can be thought as means to achieve confidentiality, integrity and availability.

Nikander lists the more detailed list of security services to be *availability, confidentiality, integrity, identification, authorization, delegation, authentication* and *non-repudiation* [51]. An additional function, which is often mentioned, is *access-control*. Using the basic services listed above, access-control is understood as a combination of identification and authentication. In the next sections we briefly describe the terms and concepts listed above.

2.1 Confidentiality

Confidentiality was the first thing addressed when talking about computer security or security in general. Confidentiality means that the information being secured in a network node is available only to legitimate users who have the right to access the information in that particular node. Thus unauthorized users will never have access to confidential data.

Cryptographic mechanisms are the most common means to ensure the confidentiality of information [30]. If we want to be sure that nobody other than us can access information we want to keep secret, we can encrypt our data using some cryptographic algorithm and store the encrypted message to our computer system. An outsider who does not know how to decrypt the data, i.e. does not know the correct decryption key, sees just gibberish.

Let us take an example from the business world. Assume that a large telecommunications company is developing a brand new mobile phone that is much smaller and has much lower power consumption than older models or any other manufacturer's models. This research and development data is crucial in the sense that if the data of the new mobile phone was stolen by some of the company's competitor, the financial loss should be huge for the company that invested in research of the new telephone. However, if the company encrypts the information using cryptographically strong methods, the loss if the data would be stolen would be zero assuming the thief cannot find out how to decrypt the stolen data.

2.2 Integrity

Although our data has been secured by encrypting, we are not finished. Consider a domain name system (DNS) as an example. In order to work properly, the DNS entries must be correct. Let us assume that someone has access to DNS entries which she is going to alter in a malicious purpose. The intruder could change the entry in a way that an IP number associated with host name, say `www.bank.com` would now be associated to the intruder's own server which is only pretending to be the actual server of the bank. In this case, the user of the bank services accessing the server through the web cannot distinguish the fake bank from the real one if the intruders server acts just like the real one. Now the intruder can collect all information the unsuspecting user gives and then use the pass-phrases or numbers to access the users account in the real bank. If tampering of the DNS tables is possible, we say that the integrity of the tables is compromised.

In practice, the integrity of information can be achieved by using one-way hash functions and message authentication codes (MAC). Hash functions will be discussed in more detail in Section 4.4.3 and MACs in Section 4.4.4.

2.3 Availability

Availability means that the network node is functioning properly, i.e. it can offer the services to the other nodes in some finite time period. The concept of availability, or of denial of service threat was defined by Virgil Gligor in 1984 [1, 49].

Until today the main security functions are considered to be confidentiality and integrity. However, computer networks are becoming a more and more essential part of large, multinational companies. For example, a company which has offices and agencies in many countries all around the world, needs a secure network connections between its offices. A situation where distant offices were unreachable due some network problems would be nearly unbearable. At least such a break would cause considerable financial losses. Thus availability has become one of the most essential aspects of security as confidentiality and integrity have previously been. However, availability is quite hard to guarantee. Protection against various kinds of attacks against availability is very hard. Such attacks could be carried out practically from anywhere. Even worse, these attacks could be distributed in such a manner that there are more than one attackers in various locations.

If we want to access, say WWW service in host `www.tml.hut.fi` we type the host name in our web browser and wait for the page to be loaded. If we are using a proxy service, the first thing is to check whether the requested pages are already in the proxy server. Basically, a proxy service is a local small cache which keeps record for the most usually requested web pages and stores a local copies of them in local hard disks. Now the response time decreases as does the required network traffic since the local copies can be used. If no proxy server is used, then the request is send to the `www.tml.hut.fi` which sends the requested pages as a reply on our request. However, if the www-server happens to be down for some reason, it cannot give us the requested service. We will not have the service we are to access and thus a denial of service (DoS) threat has happened. Why the server is down is another question. It might be a temporary maintenance break which would not last very long and is not necessarily a real DoS situation. It can also be a consequence of an attack against the server that had brought it down. In the latter case the reason for the break is a real DoS situation.

2.4 Identification

identification:	*n.* anything by which identity can be established.
identity:	*n.* the distinctive character belonging to an individual; personality; individuality. [64]

Identification means the procedures to ask the identity of a user or a network node n_1 who wishes to gain access to some other network node n_2 or service provided by the node n_2. Identification is carried out in order to ensure the confidentiality and integrity of information. When we ask the identity of a person or a node, we want to get assurance that the person making the service request has the right to access the service she wants.

Identification and authentication are usually used as a pair of concepts [1]. Sometimes they are even used overlapping one another[1]. Identification, as the dictionary entry in the beginning of this section shows, means techniques to get knowledge of the identity of another party. However, identification does not give proof for the stated identity given by the other party. As an addition, certain evidence is required to get assurance that identity presented is valid and not forged. Authentication is described as the means to verify that the person or node who presented some identification token (e.g. username) really is who she claims to be. Nikander reserves the term “authentication” for more general use, i.e. that authentication means assurance of something being authentic, and states that the identification must always be done without doubt [51].

Identification can be accomplished by three ways, with *something known, something embodied* or *something possessed* [1]. Something known means some knowledge which only one certain person knows, like password for computer system or PIN of a bank card. This means that one must never tell these secrets to anyone. Otherwise, this other person can pretend to be you and act as you in a computer system. Something embodied identification is somewhat stronger than identification based on something known, since the ability to forge something embodied is much harder. Something embodied means, for example, a persons fingerprint or retinal patterns which are both assumed to be unique for every human according to recent knowledge. One might think that a fingerprint or even a retinal patterns could be stolen by some extremely violent manner. Fortunately, the scanners can detect the life signs in a human eye [62], thus the idea of stealing an eyeball is doomed to fail. Finally, something possessed means for example conventional key, or a magnetic card or more recently, a smart card.

By combining two or three of these characteristics, we can achieve strong identification sufficient for our needs. For example, a smart card is usually combined with a PIN which must be presented when using the functionality of the smart card. In this case both something possessed and something known are used to gain more secure identification method. We will discuss more about identification in context of name certificates in Section 4.1, and in context with authorization decisions based on identification in Section 3.1 which deals with trust models.

[1]For instance, Menezes et al. [48]. defines identification by starting “*Identification* or *entity authentication*...”, i.e. using two different words to mean basically the same concept.

2.5 Authentication

authenticating: *v.t.* **1**. To make genuine, credible, or authoritative; **2**. to give legal force or validity to; **3**. to establish or certify. [64]

As we mentioned previously in Section 2.4, authentication is usually seen as part of identification [48, 70]. Traditionally this problem is solved by using username and password as pairs to identify and authenticate persons or network nodes. When user Alice wants to identify herself to Bob, she sends her username to Bob. Now Bob wants to be sure that it is “the right” Alice who introduces herself to be “Alice” instead of an imposter. Now Bob asks Alice to send her personal secret password to Bob. If Alice knows the right password, then Bob can be pretty sure that it really is Alice with whom he is communicating.

However, the problem is the password which Alice must keep secret. If she writes her password on a piece of paper for the situation she forgets it, there is a possibility that somebody finds that piece of paper and learns Alice’s password. User names and passwords are not a foolproof method to identify persons or nodes in a way that there is no place for doubt. This is why password-based identification is sometimes called *weak authentication*.

In order to gain greater assurance about the identity of the party we are identifying, more secure methods must be used. *Strong authentication* (sometimes called also *challenge-response identification*) uses a method of showing that we know a certain secret which is associated only to us without revealing the secret itself.

Nikander defines authentication to mean ways to convince oneself that electronic document or other information is created by the party who claims so [51]. When defined this way, authentication can be seen as a more general concept instead the one associated with identification.

As a real-world example we could take a purchase paid with a credit card. In order to give the vendor permission to charge our account with the given amount of money, we sign the receipt and this way we state that the credit card used to pay the purchase was really in our hands, i.e. the right owner of the card, instead of some imposter. If there is some doubt about the money transaction, the vendor can show that she has the receipt with a signature which binds the credit card used to the user of the card. If the signature in the receipt turns out to be a fraud, then we can show that someone has misused our credit card.

2.6 Authorization

authorization: *n.* The act of conferring legality. [64]

When identification deals about getting certainty who the user or network node is, i.e. to get assurance of the identity of the node or the person, authorization deals about the rights the node has. Authorization defines the rights of the user in a networked computer system, not the identity. Here the word ’user’ must be understood to mean not only a human being. A user can be a network node itself, or the person using a workstation. Common practice this far has been to base authorization decisions to the identity of the node but Nikander states that usually identity checking is not necessary [51]. On the contrary, sometimes it is not even wanted.

In the old days computers systems were most usually large mainframe systems which were composed of a one physical computer. The very first ones had no other access control than physical security. Only specially trained engineers were allowed to access the computers and execute tasks.

A more sophisticated access control mechanism compared to the physical one is the username and password which is still today used as a primary access control mechanism to show and verify an authorization to use a computer.

However, computer systems began to grow up larger and larger and were extended over a large physical area by connecting the computers together by a network so that they can communicate. A single username-password pair is not enough to express access and authorization information. A more fine-grained methodology to express authorization information is needed. Different parts of computer systems might have different security policies and usage restrictions.

As a real-world example of simple authorization, consider a police officer. When the officer is in the field doing traffic control, she has the authority to give speeding tickets if she catches somebody speeding. Here the authority to fine somebody for speeding is based on the role of a police officer. The identity of the particular police officer who gave the speeding ticket is not essential.

As the above example strongly suggests, the authorization for certain actions and rights is more relevant than the identity of the actor.

2.7 Delegation

delegation:	*n.* the act of delegating: a *delegation* of powers or authority.
delegate:	*v.t.* to commit or entrust (powers, authority, etc.) to another as an agent or representative. [64]

If a node has a certain authorization which defines the rights of that node, the node can delegate some or all of her rights to another node or person. It must be noted that it is not allowed to exceed the rights the original node possesses when it delegates those rights to another node.

When delegation of rights is done, it can be defined that the receiver of the delegated rights may or may not delegate the rights she received further on. If we use SPKI certificates [17] to the delegation, the certificate has a special entry called delegation to express whether the certificate may be re-delegated or not. SPKI certificates are discussed in more detail in Section 4.3.2.

Let us consider a key of our home as an example of delegation of rights. If we are going on a trip for a while and we want to give the key to our friend for the duration of our trip so that she can feed our cat while we are away. This is actually a delegation procedure. We delegate the right to access our home for a certain period of time to our friend. Of course, we must trust our friend to give the key back when we return. If we are using SPKI certificates, this problem will not occur, because we can assert a time period under which the certificate is valid.

2.8 Access Control

Karila states in his doctoral dissertation that access control (AC) is not a similar security function as those discussed above, but rather more involved to authentication [40]. However, Nikander's broader definition of authentication [51] suggests that access control is more involved to identification and authorization than authentication.

For example, UNIX username and password are one way to implement an access control mechanism in a distributed system. If a person has a username and password to a UNIX server, then the computer system allows her to log in. Otherwise the access is prohibited. UNIX stores the user names and password in a special file, usually `/etc/passwd`[2]. This file is an application of an access control list (ACL) which defines the valid user names for the particular computer system. If the username is in the `/etc/passwd` and the user presenting it knows the secret password attached to the username, the computer system will have a certain level of assurance that the user can be allowed to use the computer system.

Access control is usually divided to mandatory access control (MAC) and discretionary access control (DAC). Mandatory access control is a set of procedures on which the user cannot affect. Discretionary access control means such procedures which user herself can set or define. The above example of username and password is MAC type because user has no way to affect the login prompt. Instead, the computer administrator can set the MAC parameters to the system [1]. UNIX file permissions are an example of DAC. The user has in her power to set permissions as she sees to best fit her needs.

An UNIX style access control mechanism has been available and has been used since the 1960's. Therefore, a lot of scientific research has been done covering many kinds of models of access control [58, 59, 66–69]. To mention some of these models, lattices [67] and role-based [69] are some examples. However, due to the rapid growth of Internet usage and number of users, the practice has shown that ACL-based access control is no longer adequate method to manage computer

[2]Today it is more common that different variants of UNIXs uses `/etc/shadow` readable only by the root instead of `/etc/passwd` to store passwords in order to prevent different kinds of attacks like attempts to try to crack poorly chosen passwords.

resources. Firstly, in a centralized system, the identity of a user is quite well known, but this does not necessarily apply on a distributed system. Secondly, in order to have robust and scalable system, a method for delegating rights is needed. Thirdly, ACLs are not expressive enough to declare access control properties of the system [8]. Finally, different parts of a certain distributed system may have different security policies. No central policy can be applied. Recent research shows [7–11, 24, 25, 46, 54], that a more general method of resource management is needed.

2.9 Non-repudiation

Non-repudiation is a method to prevent a person or node from denying that she has sent a message or committed an action sometimes in the past. Consider that we have purchased something from a vendor as in our example in Section 2.5 discussing authentication. If we decide to deny the purchase transaction for some reason, there must be means to resolve the controversal situation. The most possible consequence is that the vendor starts a civil court case to have an independent third party to settle the controversy. A signature in the receipt is considered to be unique to every human being. In the past, a graphologist was the specialist to say whether a signature is signed by a certain person or not. Today there are more sophisticated technical ways to analyze a sample signature and the other one in the receipt and get a result whether the signatures were written by the same person.

In a distributed system, the mere knowledge that an event has taken place is not usually enough. It is often required that also the time of the event is known. The time of an event, or transaction, makes that piece of information useful to be used for non-repudiation purposes. To acquire the time of the transaction, we need a service called trusted third party (TTP) [48] which is independent from any bindings and to whom everybody trust. A TTP can be seen as a notary service of the distributed computer system. TTP signs a document or a certificate stating that the transaction has taken place at the exact moment. If there is controversy about the time of the transaction, this notary service can give an independent judgement to the problem.

2.10 Security Labels

So far we have been describing different kinds of aspects of security of a distributed computer system. The next logical step is to define some concrete tools to prevent or counter any threats, vulnerabilities, or attacks these systems have or face. One of such a system is *security labels*. To properly define this concept, we need to make some preliminary definitions which are needed to understand the concept fully.

Top Secret
Secret
Confidential
Unclassified

(a)

Restricted
Proprietary
Sensitive
Public

(b)

Figure 2.1: Security Levels

A *partial ordering* $\leq$ on a general set S is defined as a relation on $S \times S$ so that the relation is

reflexive	$\forall a \in S, a \leq a$
transitive	$\forall a, b, c \in S$, if $a \leq b$ and $b \leq c$, then $a \leq c$
antisymmetric	$\forall a, b \in S$ if $a \leq b$ and $b \leq a$ then $a = b$.

If two elements a and b are not comparable, then we say that $a \not\leq b$.

If we also require that $\forall a, b \in S$, either $a \leq b$ or $b \leq a$ holds, then set S is *a total order* [29].

Now, consider a set of security categories $C = \{a, b, c, d, e\}$. Here the partial ordering relation is the proper subset relation $\subset$. For example, let $C_1 = \{a, b, c\}$, $C_2 = \{b, c\}$, and $C_3 = \{c, d, e\}$. Now $C_2 \subset C_1$ holds but $C_3 \not\subset C_1$ and $C_1 \not\subset C_3$.

2.10.1 Security Levels

A *security level* is a hierarchical attribute that can be associated with an entity or a node of a computer system to describe the level of their sensitivity. In a distributed computer system, different parts and nodes have diverse requirements for confidentiality, integrity and availability. For example, a disk server might have higher requirements for availability than a printer server. Therefore, the security level l_{NFS} of the disk server must be set higher than the security level l_{LPD} of the printer server. Thus, $l_{LPD} \leq l_{NFS}$. Different environments of operation require different levels of security both in amount and in sensitivity. A typical examples of both military (a) and commercial (b) security levels is shown in Figure 2.1 [1], respectively. When security levels are defined in the manner described above, we can compare different levels and find out which items are in a higher level than some others. Therefore the set of security levels can be inspected as a totally ordered set.

2.10.2 Security Categories

A *security category* is a non-hierarchical set of computer system nodes which is meant to help in grouping the different nodes in different groups. A typical categorization of nodes might be according to different detachments in military environment or different departments in commercial environment. A computer node can belong to more than one category at a same time. This is one of the two main differences when compared to security levels. The other one is a straight consequence from the non-hierarchical structure of security categories, i.e. different categories cannot be arranged to any particular order according to equality, less-than, or greater-than relation. In a military environment, the need-to-know principle is usually represented by the means of security categories [1]. The need-to-know principle means that only those people who need some pieces of information to have their work done are allowed to learn it. No outsiders are allowed to acquire this piece of information because it is not relevant in their work.

2.10.3 Security Labels

A *security label* is a tag associated with every computer node which describes the sensitivity and need-to-know attributes of the node. A security label is a pair consisting of a security level and some set of security labels. Formally, a security label is a *Cartesian product* of a level and a some set of categories. For example, consider a following commercial setting[3] where we have

$$\begin{aligned} levels &= \{public, secret\} \\ categories &= \{marketing, R\&D\} \\ \mathcal{P}(categories) &= \{\emptyset, \{marketing\}, \{R\&D\}, \{marketing, R\&D\}\} \end{aligned}$$

Now the set of all security labels is a set

$$\begin{aligned} labels &= levels \times \mathcal{P}(categories) \\ &= \{(public, \emptyset), (secret, \emptyset), \\ &\quad (public, \{marketing\}), (secret, \{marketing\}), \\ &\quad \vdots \\ &\quad (public, \{marketing, R\&D\}), (secret, \{marketing, R\&D\})\} \end{aligned}$$

So far we have discussed the network nodes only as plain “nodes” without differentiating them in more detail. A very common practice is to divide nodes into active and passive ones [1, 27].

2.10.4 Subjects and Objects

A *subject* is a network node which is active i.e. it can initiate actions, make requests of resources and perform computational tasks. A subject is usually a

[3]Here the abbreviation ’R&D’ denotes the common way to shorten the ’Research and Development’ department

process on a computer system representing and performing tasks on behalf of some person. Traditionally, the relevant information of subjects that the particular computer system must know has been the identity of the user owning the process, creation time and different security attributes.

An *object* is a passive storage repository in a computer system whose main task is storing data. A typical object could be some file or a directory or maybe a whole directory tree, as well as a database. Relevant information associated to objects is owner of the particular object, size, creation time and security attributes, like read and write permissions and possible execution rights.

Usually the split in subjects and objects is quite obvious. However, in distributed computer systems the distinction becomes nontrivial. Consider a distributed computing task which divides some large problem into smaller parts and does the job in parts. In this scenario, the smaller units might send some signals of their current status to one and another or to the master node administering the distribution. Now the border between a subject and an object becomes blurred.

2.10.5 Clearances and Classifications

A *clearance* is a security label associated with a subject to denote its security sensitivity. A *classification* is a security label associated with an object in a similar manner. There is a little differences in the literature concerning this naming convention. Amoroso [1] uses terms clearance and classification as stated above to make the difference between security levels of subjects and objects. Gollmann [27] uses the general term security labels but makes the remark that sometimes the maximal security labels of the users are called clearances.

We say that a security label l_1 *dominates* a security label l_2 if and only if the security level of label l_1 is greater than the security level of label l_2 and if security category of label l_1 is a superset of security category of label l_2. Formally,

$$\begin{gathered}\forall l_1, l_2 \in labels : l_1\ dominates\ l_2 \text{ if and only if} \\ level(l_2) \leq level(l_1)\ \wedge\ cat(l_2) \subseteq cat(l_1)\end{gathered}$$

and the equality condition of a pair of security labels l_1 and l_2 is obtained by replacing the lesser-or-equal relation '$\leq$' and subset relation '$\subseteq$' symbols by equality relation symbol '$=$', respectively. It also follows from the definition that a security label l dominates itself, i.e. $l\ dominates\ l$.

2.10.6 Lattices and Hasse Diagrams

With the constructions described above, we can study the properties of security labels which form a mathematical structure called *a lattice*. A lattice is a pair $(L, \leq)$ where L is a set and $\leq$ is a partial order. A lattice has two characteristic features which are useful when inspecting the properties of a set of security labels.

We will define these two concepts in the following. If $(L, \leq)$ is a lattice, then $\exists\, u \in L$ and $\exists\, l \in L$ so that given any $a, b \in L$,

$$\begin{array}{l} a \leq u,\ b \leq u,\ \text{and} \quad \forall v \in L : (a \leq v \wedge b \leq v) \Rightarrow (u \leq v) \\ l \leq a,\ l \leq b,\ \text{and} \quad \forall k \in L : (k \leq a \wedge k \leq b) \Rightarrow (k \leq l) \end{array}$$

The element u is called *a least upper bound* and element v *a greatest lower bound* [27]. If we now want to know what is the minimal security level which a subject s must have in order to be allowed to read objects o_1 and o_2, we can find the least upper bound of these objects. Further, if we have two distinct subjects s_1 and s_2 which have different security levels or clearances, we can find the maximal classification of an object o which can be accessed by s_1 and s_2.

Returning to our set of security labels described in page 12, we can illustrate the set by a *Hasse diagram* shown in Figure 2.2.

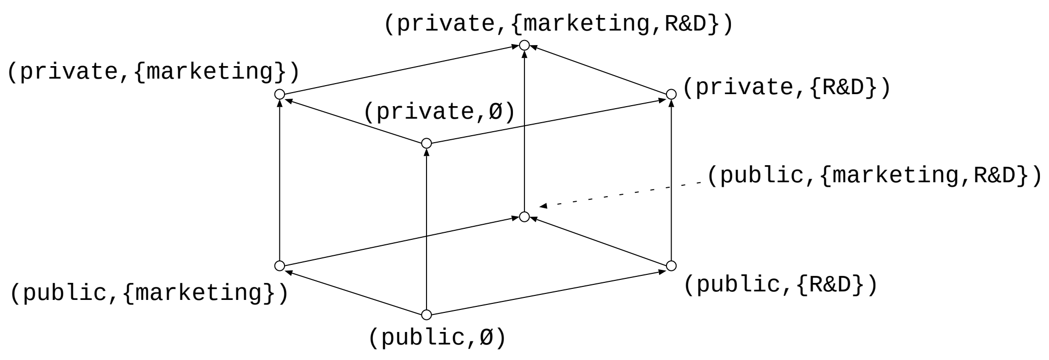


Figure 2.2: A Hasse diagram of security labels

Now we can see, for example, that following relations hold:

$$\begin{array}{rcl} (public, \emptyset) & \leq & (private, \{\text{R\&D}\}),\ \text{but} \\ (public, \{\text{R\&D}\}) & \not\leq & (private, \{\text{marketing}\}) \end{array}$$

so $(private, \{\text{R\&D}\})$ *dominates* $(public, \emptyset)$, but $(private, \{\text{marketing}\})$ does not dominate $(public, \{\text{R\&D}\})$ nor vice versa.

Chapter 3

Notion of Trust

trust: n. **1**. A confident reliance on the integrity, veracity, or justice of another; confidence; faith; also, the person or thing so so trusted. **2**. Something committed to one's care for use or safekeeping; a charge; responsibility. **3**. The state or position of one who has received an important charge. **4**. A confidence in the reliability of persons or things without careful investigation. [64]

Trust is a difficult concept to define exactly. The first thing, when thinking about trust itself as a concept, that comes to one's mind is that good friends trust each other. Friendship is usually understood as the ability to share even your deepest thoughts and feelings with somebody who understands you. This also includes the confidence that the other person will keep your secrets and does not tell them to anybody.

How do we define trust? There are a couple of different dictionary definitions of trust at the beginning of this section as defined by International Webster Dictionary [64]. Another definition that has been set says that trust is something that "*begins where prediction ends*" [41, 45]. This definition tells us that trust means something about the knowledge as well. If we had an ultimate knowledge of everything, then we would know things and we would not need to trust anything. However, since we do not have such knowledge, we have to trust instead of knowing. If we have neither prior knowledge nor understanding about some matter, then we only can make assumptions or predictions about this matter. However, if we do know something about it, we could have a bit of trust on it. Our trust on our friends is based on the knowledge of that person as a character. Usually, as is the situation when a friendship is considered, the trust is not obtained either easily or rapidly. Instead, it takes a long time to start trusting somebody (or something), it is not unusual to talk about years when building trust is considered. What it needs is a repeated series of pleasing events that enables one to build trust to somebody.

While the trust is difficult to gain, it is very easy to loose. Consider a situation when you hear some third person stating you some very personal thing, maybe something that you have kept as a secret, say, for example, that you have some kind of phobia, like agoraphobia[1]. This is, in a sense, quite an extreme example of a secret. Nevertheless, it serves as a good example of a very personal thing which usually is a big problem for a person suffering from such phobia. She might feel that the phobia is very embarrassing because other people sometimes tend to underestimate her problem by saying that "Come on, there's nothing to be afraid outside". Usually this makes the person even more embarrassed while the others does not really mean to hurt this person. If you trust your friend enough and share a secret like the one described above and next you hear your secret from another person, your trust to your friend will usually end at that very moment. It takes a *very* long time before your so called friend will gain your trust and friendship back. It is not unusual that the trusts will never be the same again.

The discussion above is a quite ordinary example of the trust as a phenomenon between humans. Is the trust in distributed systems somehow different from the trust in human relationships? First, trust can be examined in two different meanings. Usually in computer science there is an assumption of some trusted computing base (TCB), on which the theory of security sometimes heavily relies on. Trusted computing base is defined as the totality of hardware and software protection mechanisms responsible for enforcing the security policy of a given system [1, 27]. If the trusted device breaks or turns out to be not trustworthy, the whole theory usually collapses. This means, for example, that if the security of the computer system relies on the TCB, the security will be lost if the security of TCB is lost. There exists something that *has to be* trusted [52]. Trust is necessity. Secondly, the nature of trust is sometimes, like in human relationships and more psychologically emphasized context, more than in computer science, something that *can exist*. Something can be trusted, but not necessarily. Only if someone wishes to do so, like two friends do. In our previous example, the trust might be totally lost when the friend betrayed the other's trust, but before that, trust gradually evolved in a way that as time passed, there was more and more trust. However, computer cannot understand such fine nuances at all. Therefore, in a technical sense, the trust is usually something that either exists or not. There is trust or there is not, the concept is quite binary by nature [52].

The problems arise when the system of trust becomes more complicated instead of just a pair of friends as we described above. Consider a group of three people, called Alice, Bob, and Carol. Assume that Alice and Bob are good friends and they trust each other completely. Furthermore, let us assume that Bob and Carol are good friends also. They trust perfectly each other. However, nothing is said about trust between Alice and Carol. There obviously are some certain reasons connected to characters and personalities, which makes Alice and Bob trust each other. The same applies to the relationship of Bob and Carol. Alice and Carol may know each other and be good friends too, but it might as well to be the situation that they are not. They might never have met, or they have met but did not get along at all. As a summary, while Alice trusts Bob and Bob trusts Carol, this does not automatically mean that Alice would trust Carol as well, i.e. trust is not

[1]The fear of being in open or public places. Sometimes it may be so crippling illness that it prevents the person suffering from it even from leaving home.

transitive by definition. Trust is not necessarily reflexive, either. It might very well be so that while Alice trusts Bob, Bob does not trust to Alice. Further, the amount of trust need not to be equal to both directions. When a friendship is considered as in our previous example, it is reasonable to assume that the trust relation is reflexive. But in general case, the assumption of reflexivity should not be made.

As a brief summary we conclude that trust between humans is very different from the trust which a computer can process. Humans can understand the different nuances of trust. For computers, trust is more or less a binary property. It would require a somekind of an intelligence - more or less artificial - for computers to be able to process trust information as humans do.

3.1 Models of Trust

In order to express trust information in a digital world, certain simplifications need to be made. Otherwise computers cannot process that information in any sensible way. Therefore, various methods of modeling trust in digital systems have been proposed over the years. These are called *trust models*. The first, and the most simple case of trust model is *direct trust*, which, in essence means that a person trusts her own self-generated encryption key, since she knows where it came from. As a more sophisticated trust models are X.509 *hierarchical*, top-down model based on name certificates [23, 39], and the other is the *web of trust* used in PGP (Pretty Good Privacy) [86] encryption software written originally by Phil Zimmermann.

3.1.1 Hierarchical Trust Model - X.509

As the rapid growth of Internet started somewhere in the beginning of the 1990's, the need for effective directory service became evident. Figure 3.1 shows the exponential growth of Internet hosts since the beginning of the 1990's. The recent survey conducted by ISC shows the growth of the number of hosts from a little more than one million hosts (January 1993) to about 72.5 million hosts (January 2000) [38].

X.500[2] was designed to be a standard which defines one global, distributed directory of Internet users and hosts. It offers decentralized maintenance, powerful searching capabilities and single global name space among other features [82]. X.509 is a certificate standard which is part of a X.500 Directory recommendations first published in 1988. One single, global name space without the possibility to misinterpret any names was one of the greatest requirement for X.500 directory. Global name space is also a very challenging task to implement. Components of

[2]X.500 was proposed as a standard by International Telecommunication Union – Telecommunication Standardization Sector (ITU-T) and ISO/International Electrotechnical Commission (IEC)

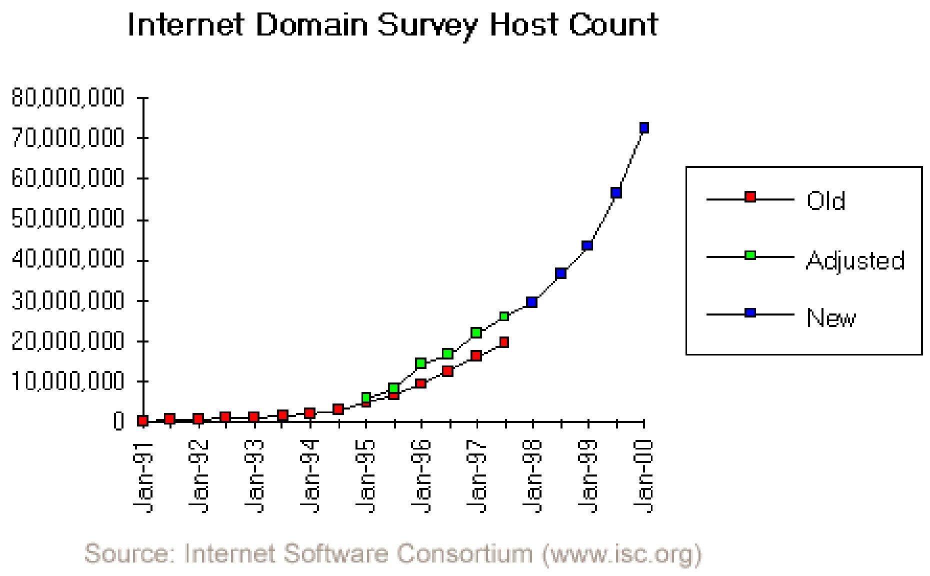


Figure 3.1: Internet host growth between 1991 - 2000 [37].

the X.500 directory entries are called *relative distinguished names* (RDNs) since the are relative to each other. When asked, we would tell that we work at

> TeSSA research project in Telecommunications Software and Multimedia Laboratory in Department of Computer Science and Engineering in Helsinki University of Technology in Espoo, Finland.

This can be expressed another way also to illustrate the hierarchical nature of our work place:

Finland
    Espoo
        Helsinki University of Technology
            Department of Computer Science and Engineering
                Telecommunications Software and Multimedia Laboratory
                    TeSSA research project

As we can see, the TeSSA project is one of the research projects in Telecommunications Software and Multimedia laboratory, which again is one of the laboratories of the the Department of Computer Science, and so on. Now, for example, we can distinguish the Dept. of CS from the department of Computer Science in University of Helsinki [28]. Generally, X.500 directory forms a tree-like hierarchy, where each node consists of several lower-level nodes.

The security the information which is stored in X.500 directories is based on X.509 *identity certificates*. We'll discuss more about the concept of identity certificate in Section 4.1, and pass it here only by mentioning it. Instead, we will now concentrate on the trust model of X.509 certificates.

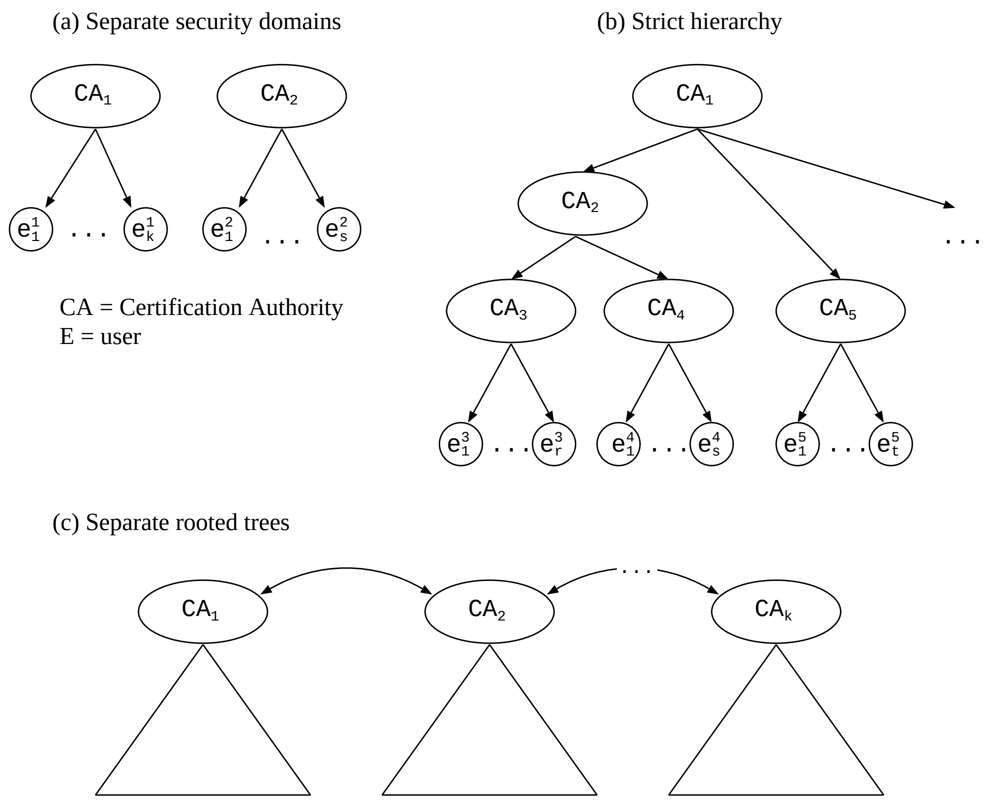


Figure 3.2: Different hierarchical trust models

As we mentioned earlier, X.509 trust model is top-down oriented. This means that there is a hierarchy of levels in which the nodes (or leaves) in a level n generally trust the nodes in the level $n + 1$ (leaf nodes are considered to be the lowest level, or level 0, and the root node is the highest level, see Figure 3.2, item (b)).

Generally, there are three different situations in hierarchical trust model [48]:

- Separate security domains
- Strict hierarchy
- Multiple rooted trees

The basic situation is depicted in Figure 3.2, item (a) where there are two distinct security domains (SD) which are administered by certification authorities CA_1 and CA_2. When user entities e^1_1 and e^1_2 wish to, for example, communicate securely, they can obtain the identity information of each other from the common trusted third party (TTP) of the single security domain, namely CA_1. All communications inside the domain controlled by CA_1 can be established easily. The problems arise when entities e^1_a and e^2_b want to communicate. They have no common trusted CA from which to get identities of each other.

To overcome this inconvenience, distinct SDs can be grouped together to form one larger SD, which is controlled by a higher-level CA. This grouping can be done again and again resulting a hierarchical tree of cascading SDs as depicted in item (b). There is always one highest 'root' certificate authority in strictly hierarchical model.

The third alternative is to group several strictly hierarchical SDs to form a forest of different security domains. In this model, there is no single highest CA, but instead, a group of peer CAs which trust each other.

X.509 trust model encompasses an assumption that by trusting the CA, everybody would implicitly trust every other CA$_n$ on which the first CA decides to trust. In X.509 model, trust is transitive in nature, which is totally different from the nature of trust between humans.

3.1.2 Web of Trust - PGP

While X.509 adopted a hierarchical model of trust, PGP (Pretty Good Privacy) adopted a totally different method of administering encryption keys and trust relations in the integrity and validity of those keys.

PGP's original creator Phil Zimmermann realized the authorities' ability to monitor and snoop - for example email - without being noticed by the original and intended sender and receiver. Concerned about the privacy of personal communication which is widely concerned as one of basic human rights [55, 73] he created a program which makes it possible for a person to encrypt and sign her emails and other electronic documents. Zimmermann saw that the new program he created could be useful for other people too. After PGP was complete in 1991, he made it publicly available on the Internet. As a consequence, he went through a three-year ordeal of criminal investigation and harassment by the US Government since cryptographic products are considered as munition and export is therefore strictly restricted [84]. US Government dropped the case in 1996 but the vast publicity of the lawsuit made people aware of the existence of PGP and therefore, it can be seen as one factor which has made PGP today the most popular encryption and privacy-protecting tool used worldwide. Since those days, the United States have made it easier to export cryptographic software [50]. Zimmermann has an impressing reputation as a prophet for freedom of speech and protection of privacy. He has received numerous both technical and humanitarian awards as well as compliments of single persons[3] of his pioneering work in cryptography [71, 75, 85]. During the recent years, new information and new disclosures about projects spying emails and telephone has been published. One of these projects which perhaps is the most widely known is called ECHELON [6, 71]. Although it is today widely

[3]On the very day in October 1993, when Boris Yeltsin was bombing the Latvian Parliament building, Zimmermann received a message from a person somewhere in Latvia. The message was as follows: "*Phil, I wish you to know: let it never be, but if dictatorship takes over Russia, your PGP is widespread from Baltic to Far east now and will help democratic people if necessary. Thanks*"

known that ECHELON is real and not just gossip talk, governments have strictly denied the existence of such projects.

In PGP, each user can create a number of different encryption/signing keys for himself. Keys can be of different lengths and the information identifying the single keypair can be chosen freely. An example of a couple of PGP keypairs is shown in Figure 3.3.

As we can see from the Figure 3.3, multiple identities can be associated to a single keypair. Although there is no specific format for the identity field, it is suggested [86] that it could be formed by concatenating a person's email address after her name, for example: `'Pekka Kanerva <pekka@tml.hut.fi>'`. Instead of email address, one could use telephone number or something unique enough to recognize the correct key so that there is no place for confusion of which key to use.

```
pub    1024   0x6AB19A96   1998-02-26   -------   DSS             Sign & Encrypt
sub    2048   0x6398DE64   1998-02-26   -------   Diffie-Hellman
uid    Pekka Kanerva <pekka.kanerva@hut.fi>
uid    Pekka Kanerva <ptkanerv@niksula.cs.hut.fi>
uid    Pekka Kanerva <ptkanerv@cc.hut.fi>

sec+   1024   0x5A31BA34   1998-11-25   -------   DSS             Sign & Encrypt
sub    2048   0x5F1774F9   1998-11-25   -------   Diffie-Hellman
uid    Pekka Kanerva <pekka.kanerva@tml.hut.fi>
uid    Pekka Kanerva <Pekka.Kanerva@tcm.hut.fi>
uid    Pekka Kanerva <pekka@tcm.hut.fi>
uid    Pekka Kanerva <Pekka.Kanerva@hut.fi>
```

Figure 3.3: An Example of PGP keys

Keypairs are stored in *keyrings*, which are essentially a couple of files, one for public keys and one for secret keys[4], stored in user's home directory. Public keyring contains the public keys of one user as well as public keys of other users which have been received sometimes in the past from a keyserver or directly from another user, maybe in a floppy disc or attached to an email.

Figure 3.4 shows an example of a generalized illustration of the PGP trust model with some fictional users. Different ellipses represents different users and arrows denotes the public keys stored in public keyrings of users. For example, there are arrows from Alice to Bob, and David, so Alice has the public keys of Bob and David stored in her keyring. We assume that each user has made sure that the keys stored in their keyrings really belongs to right persons and that the keys are valid.

As PGP does not have any single highest root authority who certifies all the keys, everybody can act as a certifier. Using our previous example of friends, Alice and Bob can express their trust on each others public key by signing the public key and associating a trust level to the signature. PGP has three levels of trust [4],

[4]There is no limitation to the number of key files, one user can have several of them, but if not otherwise configured, PGP will use the default files `pubring.pkr` and `secring.skr`.

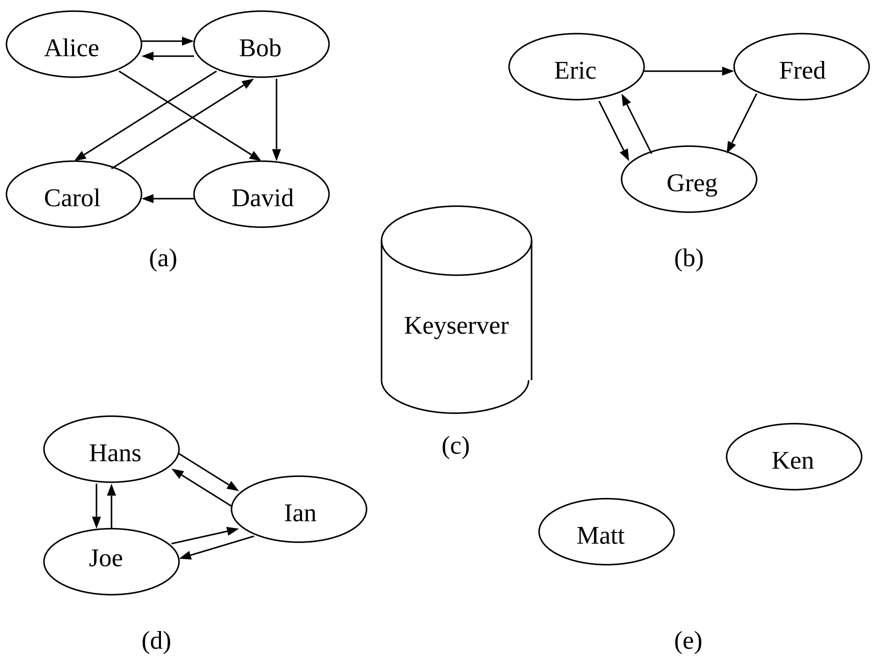


Figure 3.4: PGP's web of trust

complete, *marginal* and *untrusted*, one of which can be stated when signing somebody's public key. The situation is the same in the case of Bob and Carol. Now, if Alice and Carol want to send secure email to each other, they first need to have public keys of each other. The situation can be seen in Figure 3.4, item (a). Now Bob can act as a *trusted introducer*, who has public keys of both Alice and Carol. Since both Alice and Carol trust Bob, they can be quite sure that Bob would not fool them by giving false public keys instead the real key of Alice and Carol. When Alice and Carol get the public keys of each other, they can check whether Bob has signed the keys to certify their validity. Now, if Alice and Carol wish, they can sign the keys to certify that they also trust and believe that the keys are valid.

The various small groups of principals in Figure 3.4 shows the unstructured nature of PGP trust model. In general, trust relations form a directed graph. These graphs can vary considerably depending on whose graph is in concern. For example, in item (b) Eric has the public key of Fred but not vice versa. This can be the situation, for example, in which Eric sends regularly some report to Fred.

Figure 3.4 item (d) shows the situation where everyone of the local PGP user group have each other's public keys and thus can always encrypt their message transfer. Item (e) shows the situation of single users Ken and Matt, who do not have any foreign keys in their public keyrings. For example, this can be the situation where these persons just uses PGP to encrypt their private files stored in their workstation's hard disk, but who does not use PGP for encrypting emails.

Problems arise, when two distinct users want to exchange public keys but, firstly, they does not know each other beforehand. Secondly, they cannot meet each other since they, for example, live far away from each other. As a solution to this problem, PGP introduces a *keyserver* which simply receives and stores public keys users send to it (a keyserver is shown in Figure 3.4, item (c)). How can we be sure that we will get the right public key from the keyserver?

Consider a situation where Alice wants to send an encrypted message to Ken. First, Alice downloads the public key of Ken from Keyserver. Then, Alice encrypts her message with this public key and sends it to Ken by email. Unfortunately, a malicious imposter Matt has generated a public key with identity of Ken and sent it to the Keyserver. This is possible, because the Keyserver cannot know who is the sender of the public key and whether she is the legitimate owner of the public key. Now Matt can receive and decrypt mails sent to Ken because Matt has the matching private key. Matt even can have the *real* public key of Ken and use it to re-encrypt the messages originally intended for Ken to receive and send them to him so that nobody would suspect any wrongdoing. Of course, Ken would (or at least he should!) wonder why Alice has not signed her message. Matt cannot generate a false signature of Alice provided that he has not forged another key-pair with ID of Alice. Furthermore, Matt can even forge signatures of Ken, since everybody who wishes to verify the signature would use the bogus public key of Ken. Digital signatures are discussed in more depth in Section 4.4.2.

How can we prevent this disaster from happening? The PGP way to do this is to ask somebody to validate the public key before it is sent to the Keyserver [4]. Ken can ask some of his friends to sign his public key and thereby vouch for the integrity of the public key. Although Alice does not know Ken nor any of his friends, she can check the public key she downloaded from the Keyserver and see that *somebody* has signed Ken's public key and therefore Alice can lay a bit more trust on the validity of the public key compared to the situation that the key would have no signatures at all. A much better situation would be, if Alice and Ken would have someone as a common friend, who could act as an introducer between them.

As a final note of the trust model used in PGP is, that although PGP does not enforce any structured trust hierarchy, it works equally well in an environment with some centralized Certification Authority.

3.2 Trust Management

As described above, the two major existing concepts for expressing trust relations in distributed network, namely the X.509 and PGP, address only a portion of a general problem of *trust management*. X.509 solves the problem by introducing an optimistic, general model of hierarchy where everyone implicitly trust some CA who is certifying that the mapping from a person's name to some public encryption key is valid. PGP took an opposite direction on deciding whom to trust. The starting-point [4] for Phil Zimmermann was to preserve personal privacy against governmental projects aimed to give authorities practically free abilities to listen and tap personal communications between ordinary citizens. Because of the baseline, the user of PGP can choose freely the trusted parties by giving public keys some statements about their trustworthiness. The documentation enclosed in the distribution of *PGPfreeware* [4] must be seen as a scientific study. Instead, it is a more commercial advertisement which emphasizes the benefits of the PGP, as in pointing out that PGP certificate model supports both the web-of-trust based solution and hierarchy of CAs as well. X.509 as well as PGP solves only a smaller subproblem of trust management.

Until today, authorization questions have been answered by dividing the question into two different pieces, namely to *authentication* and *access control*, which are handled differently. For example, when logging into a multi-user computer environment, firstly, the computer asks for a username and a secret password to authenticate the user[5]. After successful login, user is free to use the computer in the limits set by computer administrator. As we mentioned before, for example, in UNIX machines, users may set various permissions for her files. These permissions control which users can read, write or execute her files. When another user tries to read, write, or execute those files, operating system makes a check by matching the username making the file access operation to the file permissions of the file in question and makes the access control decision.

Centralized model of access control has worked pretty well in centralized computer systems or even in distributed systems, which are closed or relatively small in size measured by number of user accounts. UNIX systems, for example, have been serving computer users a long time since the days of its creation in 1969. A good example of a quite large-scale computer facilities is the Computing Center of Helsinki University of Technology. In fall 2000, the total number of users in the UNIX system available for both HUT staff and students is roughly 15,500 users[6].

However, since Internet has been growing exponentially whether measured by the number of hosts (see Figure 3.1) or by the number of users, computer systems are not so centralized anymore nor small in size. Therefore, as the recent studies show [7, 8, 10, 11, 24, 25, 46] a more general method of trust management is needed. Consider a www-based service where the subscribers of a newspaper, for example Helsingin Sanomat, can browse the electronic copy of the newspaper in Internet. In this context, it is not necessary nor required to know who the person browsing the paper is. The service provider knows the address of the electronic subscriber, i.e. the IP number of the subscriber so there is even no need to resolve the address of the user by finding out the username. The valid question is, whether

[5]Actually, to be precise, logging in to a computer is also partly access control of nature. If the username the user presented is not listed as a valid username for the system in question, the access is forbidden.

[6]Measured in October 19, 2000 by counting entries in `/etc/passwd` in computer `kosh.hut.fi`. We executed '`cat /etc/passwd | wc -l`' which gave the exact count to be 15,404 users. 2,301 of them had the shell set to `/bin/lupailmo`, i.e. the account is closed.

the electronic subscriber is also a subscriber of the conventional paper, i.e. "*Is the requester authorized to perform the action?*" The fundamental reasons, which renders traditional ACLs inadequate [8] in distributed systems are:

authentication in an operating system, the identity of a principal is usually very well known which does not apply in distributed systems.

delegation is necessary to achieve better scalability of the distributed system. For example, administration can be distributed with the help of delegation.

expressibility and extensibility. Usually ACLs are implemented as a part of some application program or an operating system. This sets certain limitations to the implementation, which, in turn, sets constraints to the extensibility.

local trust policy As computer systems grow large, different parts of the systems may have different needs for security. Therefore, no global security policy can be applied.

Generally, a trust management system unifies the notions of security policy, credentials, access control, and authorization [7]. The base of the trust management is the *security policy* which defines the conditions and rules under which the service is provided. When an user *requests* the service, she presents a set of *credentials* to prove that she has the right to use the provided service. The algorithmic core of the trust management is an engine which performs the *compliance-checking* and decides whether the service should be provided or not. Formally, given some request r with a set of credentials C and a local security policy P, does the set C prove that the r complies with P? Thus, the trust-management engine is a function $f(x_1, x_2, x_3)$ which has the value true, if r and set C comply with the local security policy P, or false otherwise. i.e. $f(r, C, P) \rightarrow \{\mathsf{true}, \mathsf{false}\}$ [10].

To achieve general, scalable, and flexible trust management system, it is crucial that the compliance-checking engine is implemented independently from any application [9, 11]. As in general level, the problem of *proof of compliance* (POC) is anything but a trivial question to answer, and is undecidable without any limitations. Various limitations can be stated for the compliance checking procedure, for example, limiting the global or local runtime. Even with these limitations, the bounded POC is computationally intractable [11]. This means, that implementing a compliance checker is very challenging task, and it should be done with great care so that the resulting checker can be proven to be sound and reliable for both the design and the implementation phase. This is very important thing since applications using a standard compliance checker can be sure that the output of the compliance checker depends only on the input, i.e. the the request, the set of credentials and the policy, and not on any feature or a real bug in the design or implementation of a self-made checker.

From recent past we know several incidents of how lousy and security-ignorant programming and software design could cause even large corporate networks to crash. Some recent cases which got a lot of publicity were email worms Melissa [56, 76] and LOVELETTER [5, 33, 61]. These email caused enormous amounts of email traffic which jammed the local networks totally. LOVELETTER even destroyed data files from workstations. Nonetheless, it is not even enough to do to check that the system implementation complies the specifications. A few years past, an attack called ’Ping-of-Death’ [12] crashed not only workstations with operating systems but also firmware like routers, and peripheral devices like printers and dumb terminals. The attack was carried out by sending an over-sized `ping(1)` packet[7] which usually caused the destination node to hang, kernel panic or reboot.

A compliance checker programmed in a similar fashion like the the aforementioned email software or office suites would be no good. The ’Ping-of-Death’-example shows that even meeting the requirements of the specifications might not be enough. Even more, the programmers must consider also the exceptional situations outside the specifications and be prepared to handle those cases also. Lousy work done by somebody else could jeopardize the stability and integrity of our compliance checker[8]. As policies and credentials become more complex, the role of good compliance checker grows even more. A clean separation of the compliance checker and applications using it makes the whole system more modular and therefore more manageable and easy to customize or tailor to various needs.

[7]The ICMP ECHO request which is used by `ping(1)` is normally 64 bytes of size. Some implementations of `ping(1)` (Windows 95, Windows NT 3.51 and NT 4) allows user to set the size of the `ping(1)` packet freely. Usually these over-sized packets need to be fragmented to several smaller packets which are transmitted over the network. The fragmentation is needed due the maximum packet size of the IP protocol or due the maximum transfer unit (MTU) of the underlying network. The packet reassembly in the destination host causes an unexpected condition which in turn can cause an overflow of 16 internal variables. The case was special because of the peripherals and firmware were in danger also.

[8]Very often the attacks against computer systems are based on some misuse of a certain protocol or such. It is *very* hard to come up with *all* possible ways to misuse and to protect the system against them all.

Chapter 4

Digital Certificates

certificate: n. **1**. A written declaration or testimonial. **2**. A writing signed and legally authenticated. [64]

Certificates play an invisible but important role in many essential episodes of life. We all have birth certificates, if we get married we will have a marriage certificate. If somebody gives us a valuable gift we might also have a gift certificate which may be needed for taxation purposes. Finally, one day when we depart this world we will have death certificate. As we now see, a certificate means a document of proof or evidence of some matter or event which is sometimes also legally important. In some cases there are a public notary who signs and thereby proofs the legality of the signed document. Some other cases there are two independent persons acting as witnesses for the action.

Digital certificates are quite similar by nature. The very first function of a digital certificate was to be a signed proof given by some trusted Certificate Authority (CA) which proves that some public encryption key k^+ belongs to some person. Some literature still defines digital certificates only as a certified binding between some name (or other identifying thing) and a public key[1]. We want to emphasize the more broad definition of a digital certificate. E.g. Ellison et. al. [19] define a certificate as "*a signed instrument that empowers the subject*". Thus, it follows that a certificate always contains at least two objects, an issuer of some property and a subject receiving that property. Issuer is the signer of the certificate and the source of the empowerment. Subject is the thing which is empowered by the certificate. As a general level, this 'thing' can be a name (like in identity certificates), or some object, or a hash of an object. Certificates can further be categorized in three groups - *identity certificates* (sometimes called also name certificates), *authorization certificates,* and *attribute certificates* We will discuss each one of these in more detail in the following sections.

[1]For example, Feghhi, Feghhi and Williams [23] still define digital certificates to mean only bindings of a name and a public encryption key.

4.1 Name Certificates

As mentioned before, the very first function of a certificate was bind some name to a public key, i.e. to make the mapping $(\mathsf{name} \rightarrow \mathsf{key})$. This mapping should be done by some trusted party, which is called *a certification authority*. In particular, the X.509 identity certificates does the mapping of some *distinguished name* (DN) to the corresponding public key $(\mathsf{DN} \rightarrow \mathsf{key})$.

4.1.1 Certification Authority

A certification authority (CA) is a trusted organization which accepts certificate applications from people or organizations and authenticates those applications. After that, the CA issues certificates to the applier and keeps list of valid certificates. The most important phase is the confirmation of the identity information the certificate applier presents to the CA [23]. If the data were incorrect, the CA must notice the error (and the possible imposter falsifying identity information) before issuing a certificate. Otherwise the imposter can essentially masquerade to some other person and carry out frauds.

The conditions and requirements and policies under which a certain CA operates, can be provided in *a certificate practice statement* (CPS). CPS is a free-formed document describing, for example, the liability issues and the usage of certificates the CA issues. It can be anything from a few pages to over a hundred pages long document like the VeriSign CPS^{TM2} is. Recently, VeriSign released a security alert concerning two certificates which were issued to a person who fraudently claimed to be a representative of Microsoft Corporation [77].

After the certificate is issued, the CA must keep a record of the validity of the issued certificates. By default, a certificate has a validity period after which it becomes invalid. But a certificate might become invalid for some other reasons also. The private part of the certified key might become compromised for some reason. A compromised key cannot be used anymore and therefore the whole certificate must be revoked as soon as possible. Therefore, the CA publishes regularly *a certificate revocation list* (CRL) which lists the *revoked* certificates which cannot be safely trusted or used anymore. It depends on the CA how often it publishes CRLs. A typical period might vary from hours to weeks, depending how critical the validity information of the certificates is considered. CRLs can be distributed over untrusted networks, since they are signed by the CA. If somebody tampers with the CRL, end-user can detect it by verifying the signature of the CRL.

CRLs can be distributed to the end-users in three different methods [23]. We will describe each of these methods shortly. An application can *poll for CRLs* periodically. It is required that the application knows the updating interval of the CRL so it knows when to poll for a new CRL. Between the updates, the application

[2] https://www.verisign.com/repository/CPS1.2/CPS1.2.pdf

is operating in a somewhat unsafe manner because it is unaware about the possible certificate revocations which could have happened after the last poll.

CA can *push CRLs* every time when a revocation of a single certificate has taken place. This ensures that the applications are always working with a fresh and up-to-date CRLs. The drawback is that if an intruder manages to destroy the CRL during the network-transmission, the applications will not receive the revocation information. Another problem is network traffic which could congest the network if revocations would happen frequently.

The third way is to do *on-line status checks* for the CA to obtain the validity information. This method has multiple advantages. First, it avoids the unsure periods of operation between the CRL polls. Secondly, the network traffic is much smaller because the whole CRLs are not broadcasted to everyone who might need them. Further, the CA does not need to keep a centralized database of the CRL users because the end-users initiates the queries. However, the on-line check requires that the CA is available all the time which makes it vulnerable to DoS attacks. Single queries must also be signed by CA which might cause heavy load to the CA's server if queries come frequently.

4.2 Attribute Certificates

An Attribute Certificate (AC) makes a binding between some attribute and a DN ($\mathsf{attribute} \to \mathsf{DN}$). An attribute could be some authorization or a membership information about the DN belonging to some group. One example could be an attribute certificate which states that some DN is a doctor. Thus, she has the right to practice medicine - to fill prescriptions, to perform surgery and so on. Thus, the difference to the identity certificates is that instead of some public key k^+, we have an attribute or a list of attributes bound to some DN.

There is only few studies available discussing attribute certificates. The concept of an attribute certificates is defined in American National Standards Institute (ANSI) standard X9.57. The PKIX working group[3] of The Internet Engineering Task Force (IETF) has noticed the need to define and support attribute certificates in X.509-based PKI. The main points of their proposal [22] are:

1. **The different nature of PKCs and ACs**. Public key certificate (PKC) binds a DN to a key as noted before whereas AC binds a set of attributes to a DN. PKC can be thought as a passport which identifies the keyholder and AC as a visa. It's usually needed to present a valid passport to get a visa.

2. **Different life-cycles**. DN-public key mapping can last quite long, as in the passport metaphor, the lifetime can be months or even years. This is

[3]Public-Key Infrastructure (x.509) (PKIX) working group's home page can be found in `http://www2.ietf.org/html.charters/pkix-charter.html`. The goals of the working groups is to develop Internet standards to utilize X.509-based PKI

the opposite to authorization information which might be valid only a very short period of time, maybe only for some hours.

3. **Different sources**. Different authorities issue PKCs and ACs. Some governmental authority can issue identity certificates after a rather long period of validating the information the party acquiring the PKC presented. Thus, a PKC is rather hard to obtain, since the information must be checked to be correct to prevent frauds. A completely different authority or even several authorities can issue different kinds of ACs depending on the attributes or authorizations they want to give a person.

4. **Identity is not essential**. There are a lot of situations where the identity of DN is not the critical information when the access control decision is made. The role or group information to which the DN belongs is more important.

Ellison et al. present similar results in their work on simple public key infrastructure (SPKI) working group [19]. The public key is the most usual way to act and be recognized safely in computer networks. Therefore the verifier of some request must make two distinct mapping in order to verify the request, i.e. first, to find out and check the $(\mathsf{attribute} \to \mathsf{DN})$-mapping and secondly, the $(\mathsf{DN} \to \mathsf{key})$ to have the full $(\mathsf{attribute} \to \mathsf{DN} \to \mathsf{key})$ mapping [19]. Because the two different certificates usually come from distinct issuers, both must be trusted separately which raises the security requirements of the whole process.

ACLs can be actually seen as a form of an AC. Traditionally, the function of an ACL is to list some kind of properties or authorizations to different names. From this point of view, attribute certificates have quite thorough research being done as we mentioned in Section 2.8.

4.3 Authorization Certificates

Authorization certificates are designed to make the mapping $(\mathsf{key} \to \mathsf{authorization})$ in a one, single certificate. The need for simple authorization certificates arose from the difficulty to give unique names to different principals in a distributed computer system [19, 54]. The SPKI working group has come up with a solution that a single, distinguished X.500-like name space is not very likely to occur ever. We will discuss the grounds of this argument in more detail in the following section.

4.3.1 Global vs. Local Name Spaces

The SPKI working group proposes the application of *local name spaces* which was originally proposed in Butler and Lampson in the A Simple Distributed Security Infrastructure (SDSI) [65] research project. In a small environment, the simple mapping of names to public encryption keys makes sense. To use names as identifiers is a natural way for humans which everybody has learned in childhood. In the past, people used to live in small communities where almost everybody knew each other. In such a community to know a person was equal to know the special characteristics of that person. Furthermore, it was also equal to know the name of the person and to know his identity [16]. This doesn't apply any more. Telephones, faxes and more recently, Internet has made the globe smaller in the sense that we can have conversation with other people who are only characterized by some voice in a telephone or some email address in our email program. There is no such thing as a global distinguished name. The SPKI working group have listed several reasons which strongly argue against a successful creation of any global name space in the future:

- Computers do never make any decisions based only on a name. There is also some additional data on which the decision is based.
- The names we use to identify people are unique only on our own domain, but not globally.
- A global database which would contain enough information for making some security decisions would be such a big security violation that it would be rejected as politically impossible.
- We do not necessarily know the whole name of the persons we are dealing with.
- When searching manually some database, we might pay attention only to the name of a person and ignore the other part of the DN and therefore make a mistake.

Yet another argument for justifying the global DNs is to have an *inescapable identifier* which would restrain a person from doing any evil under some name [19]. After a misuse - a crime or such - the person could change her name and start over again. Since there are several companies doing identity certificate business, there is no way to hold a person to change the CA in the fly to some another with another DN's name space. So the inescapable identifier created by a CA is not so inescapable after all.

The SDSI 2.0 method of describing local names is to use a reserved word `'name'` and the name of the principal, for example,

```
Alice:  (name Bob)
```

represents a simple name 'Bob' in Alice's name space. Further, if Bob defines a name of his own, like,

```
Bob:  (name Carol)
```

then Alice can refer to this same entity by stating

```
Alice:  (name Bob Carol)
```

Thus, the local name spaces can be used not only locally, but also globally [19].

4.3.2 SPKI Certificates

An SPKI certificate is a result of work done by the SPKI working group. An SPKI certificate is designed to authorize some action, grant one or more permissions or capabilities to a keyholder. The identifying thing here is the public encryption key, which can be assumed to be unique. Instead of the public key itself, a hash of a key can be used instead. We will discuss hashes and hash functions in more detail in Section 4.4.3.

Definition 4.3.1 *An SPKI certificate [18, 19] is a five-tuple* $(\mathcal{I}, \mathcal{S}, \mathcal{D}, \mathcal{A}, \mathcal{V})$*, where*

- **Issuer** $\mathcal{I}$ *is the principal issuing the certificate and granting the permissions or rights it communicates*
- **Subject** $\mathcal{S}$ *is the principal or name acquiring the permission or rights*
- **Delegation** $\mathcal{D} = \{\mathsf{true}, \mathsf{false}\}$ *is a flag noting whether the subject* $\mathcal{S}$ *has the right to delegate all or part of the permissions it acquires through the certificate*
- **Authorization** $\mathcal{A}$ *is a field specifying the permission being communicated*
- **Validity** $\mathcal{V}$ *is the specification of the dates or on-line conditions under which the certificate is assumed to be valid.*

SPKI certificates adopt a totally different view point compared to X.509 certificates. SPKI certificate specifications does not give any definitions how to express the authorization rules. The representation format is left to the author of the application software, so she can tailor it to best suit to the application-specific needs. Similarly, the distribution of the SPKI certificates is left as an open question. The working group have made an assumption that the certificate distribution is made directly from the issuer $\mathcal{I}$ to the verifier so no special distribution infrastructure is needed. If somebody wishes to use some directory services like LDAP [80], DNS [15, 31], or PGP keyserver, the specification and implementation is again up to the application author. Further, the SPKI certificate is expected to carry only the minimum information necessary to get the authorization decision done so that the risks to any security or privacy violation would be minimal. Also, SPKI certificates must support anonymous certifying. Anonymity of SPKI certificates is considered very important property, since there are many different kinds of applications to certificates where anonymity is highly desirable or even necessary. Some applications requiring anonymity are secret balloting, elections, and auctions.

4.3.3 TeSSA

TeSSA (Telecommunications Software Security Architecture) is an architectural model [54] which puts the SPKI certificates into use as conveying medium of the authorization and delegation information. Figure 4.1 depicts the different conceptual building blocks of the TeSSA architecture [54]. The original definition of the SPKI certificates in the request for comments (RFC) [19] does not specify where the certificates should be stored. In TeSSA, the certificates are stored in the domain name system (DNS) [31].

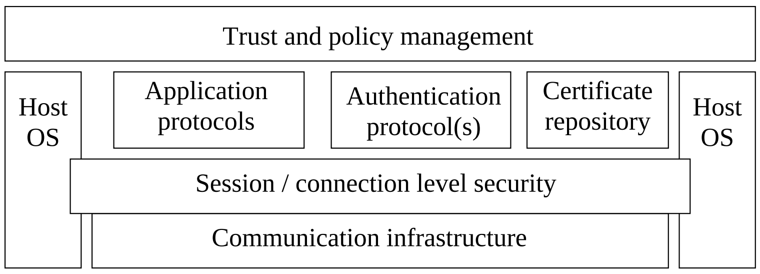

Trust and policy management
Host OS
Application protocols
Authentication protocol(s)
Certificate repository
Host OS
Session / connection level security
Communication infrastructure

Figure 4.1: Telecommunication Software Security Architecture (TeSSA) conceptual building blocks

All communications is based on the TCP/IP protocol suite which is the family of protocols on which the Internet is based. Since IP which is the protocol responsible for only transmitting the datagrams in order over the network [63] provides no security, another protocol for ensuring secure connections is needed. The Security Architecture for the Internet Protocol (IPSEC) [42] is an IETF standard for securing IP packets. The usage of IPSEC is optional in current IPv4 implementation but it is a mandatory part of forthcoming IPv6 [14]. The handshaking and negotiation of encryption keys and such needed to establish secure communications channel is done by the Internet Security Association and Key Management Protocol (ISAKMP) [47]. Java Virtual Machine (JVM) is operating as the host OS in the network nodes. JVM was chosen because it offered object-oriented environment with decent access control characteristics.

In TeSSA architecture, anybody can act as a CA and issue certificates of her own. This has been seen wise choice for several reasons. Firstly, requesting a name certificate from some CA usually takes quite long time, at least a few days or so. Secondly, name certificates are valid usually for a long time, at least weeks or months. The long-liveness property itself is not a problem. A typical example of a name certificate is an SSL certificate which is acquired from some CA to a WWW server. It is reasonable that such a name certificate is valid for some months or even a whole year. A WWW server is expected to work a long time and it is not sensible to issue certificates with validity of a few days only. The SSL certificate of some web server might become invalid by a corporate bankruptcy or a corporate acquisition. Combined with the long acquisition time, the long lifetime of a name certificate is not a problem. Thirdly, the most usual situation is to authorize some action, which is slightly cumbersome usage for name certificates in the first place. Instead, an architecture of SPKI-certificates is proposed where the certificates have relatively a short life span but which allows more fine-grained management of trust.

Because SPKI certificates allow delegation of rights and capabilities, certificates form chains where certificates are semantically bound together.

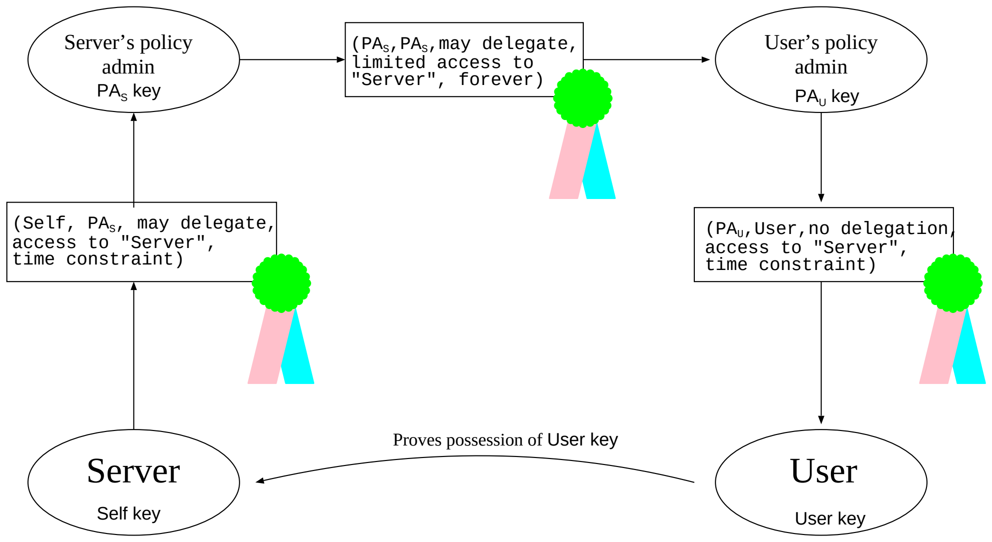

Server's policy admin
PA$_S$ key
(PA$_S$,PA$_S$,may delegate, limited access to "Server", forever)
User's policy admin
PA$_U$ key
(Self, PA$_S$, may delegate, access to "Server", time constraint)
(PA$_U$,User,no delegation, access to "Server", time constraint)
Proves possession of User key
Server
Self key
User
User key

Figure 4.2: Basic authorization certificate loop

Figure 4.2 shows a possible authorization certificate loop with TeSSA architecture. The Server has delegated the permission to access the service to its policy administrator (PA). The delegation bit in this certificate is set on so that the PA can further delegate the access. The access authorization propagates to the end-user who proves to the server that she is the owner of the private part of the user key, which closes the certificate loop.

Another certificate loop is needed for the user to be sure that she is accessing the correct server and not an imposter. Figure 4.3 shows the identification loop. The CAs need not to be official CAs like the ones we discussed in the case of X.509 name certificates. The assurance of the identity of the server can be obtained via more unofficial path.

For example, Alice can ask Bob to water her house plants while she is on vacation. Alice has a modern electronic lock in her front door, so she can authorize Bob to access her apartment to water the plants by issuing Bob a certificate which is valid during her vacation and which gives Bob permission to open the front door of Alice's home. Because Alice has signed the certificate with her private key, the electronic lock in her door can verify the signature of the certificate and deduce that the incomer is in a legal business since he just presented a certificate which is signed by Alice.

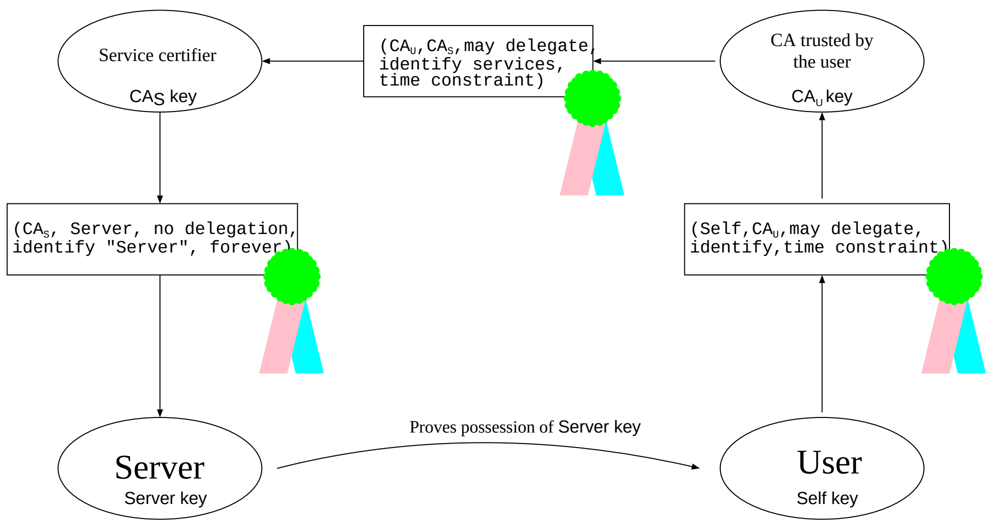

Service certifier
CAS key
(CAU,CAS,may delegate, identify services, time constraint)
CA trusted by the user
CAU key
(CAS, Server, no delegation, identify "Server", forever)
(Self,CAU,may delegate, identify,time constraint)
Proves possession of Server key
Server
Server key
User
Self key

Figure 4.3: Basic service identification loop

The advantages one achieves by using TeSSA architecture is that no physical keys have to be given to Bob. Further, the key in sort of self-degrades, since the certificate is valid only a limited time interval. Thus, there is no need for Alice to collect a key back after her vacation since the certificate which is acting as a key will be invalid after the predefined time period.

4.4 Cryptology used in certificates

Since the certificates are transferred over an untrusted network, they must be protected from tampering and alteration. The most usual situation is that the issuer of the certificate signs it to ensure the integrity of the certificate. If the certificate is altered during the transmission, the receiver or the verifier notices the alteration when she checks the signature.

We will discuss the cryptographic primitives which are needed to provide the needed security features for transmitting certificates safely over an untrusted network.

4.4.1 Symmetric and asymmetric cryptosystems

A cryptosystem in general is a tuple $<\mathcal{P},\mathcal{C},\mathcal{E},\mathcal{D},\mathcal{K}>$ where $\mathcal{P}$ is a set of all possible plain texts, $\mathcal{C}$ is a set of all possible cipher texts, $\mathcal{E}$ is an encryption transformation, $\mathcal{D}$ is a decryption transformation and $\mathcal{K}$ is the set of all possible keys.

Cryptosystems are in an essential role when building secure communications systems. Symmetric cryptosystems are much older and hence much more thoroughly studied. Yet they have an inconvenient property, namely the need to share a secret key or password, before the cryptosystem can be established. Figure 4.4 shows the principle of a symmetric cryptosystem. The name symmetric comes from the fact that the same secret key is used both to encrypt the message and to decrypt the ciphertext to reveal the original message.

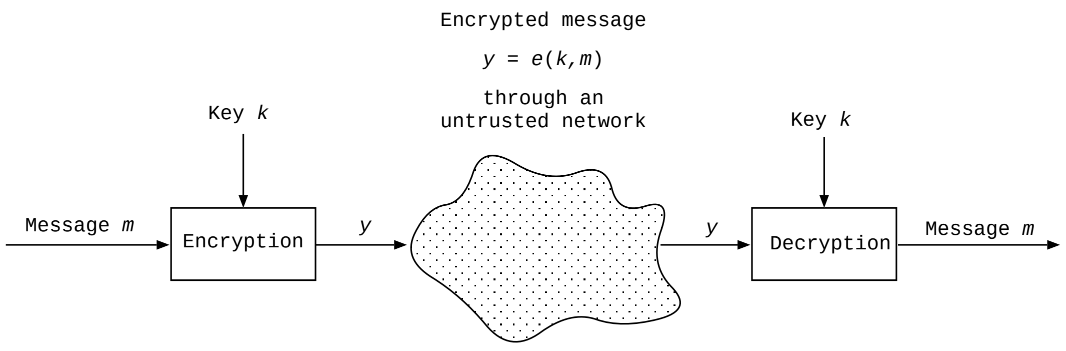


Figure 4.4: Symmetric cryptosystem

Since the key must be shared and yet secret, there must be some secure way to share the key beforehand. In the old times when the cryptosystems were relatively simple and the pass phrases or passwords much shorter than today, the sufficient condition was that the parties could meet in public and share the password. As a very simple example, consider a cryptosystem where we take some single word as a password and then list the remaining letters in order. The only requirement for the password is that every single letter appears at most one time in the word. For example, the word `'hauptwerk'` would be such a word[4].

plaintext:	`abcdefghijklmnopqrstuvwxyz`
ciphertext:	`hauptwerkbcdfgijlmnoqsvxyz`

Figure 4.5: A simple substitution cipher

Figure 4.5 shows the encryption transformation for such a substitution cipher. The system works as follows. In order to encrypt a word, say ’cat’, we start to replace letters by taking the plaintext letter and checking from the table which is the corresponding ciphertext equvalent. For letter ’c’ the cipher letter is ’u’, and continuing this way we obtain the ciphertext ’uho’. The decryption transformation can be obtained by inverting the encryption transformation.

[4]When choosing a password for a substitution cipher like this one, the great advantage is to choose an uncommon and perhaps a foreign word which is harder for an intruder to guess. The reader must also note that the cryptosystem presented above might have provided some security some hundred years ago but can be trivially cracked by computers today. Also, statistical analysis of the cipher helps to crack it quite effectively [72].

Figure 4.6 shows the principle of an asymmetric or public key cryptosystem. As we described in the Section 3.1.2 when we discussed PGP, an asymmetric cryptosystem consists of two keys, a public and a secret one which is often considered a pair[5].

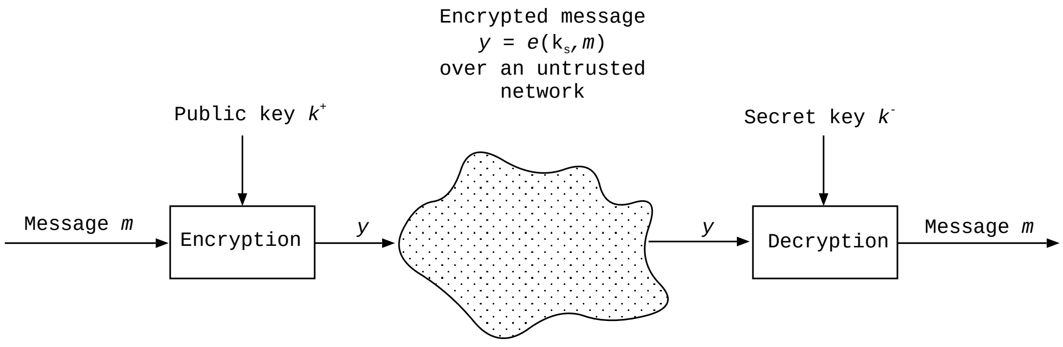


Figure 4.6: Asymmetric cryptosystem

In a public key cryptosystem everybody has a public key which can be published in a newspaper or in a public directory or such. If Alice wants to send an encrypted message to Bob using a public key cryptosystem to encrypt her message, she must first obtain Bob's public key. For now we assume that both Alice and Bob will somehow obtain the correct public keys of each other via some secure channel.

The system works as follows: Alice writes a message m and encrypts it using Bob's public key k^+_{Bob} to have an encrypted message $c = e(k^+_{Bob}, m)$ where c is the encrypted message, and $e = e(k, m)$ is a encryption transformation of two variables, namely the public key k and a message m. Now Alice can send the encrypted message c to Bob, who in his turn deciphers the message with his secret key using the decryption transformation to obtain the original message $m = d(k^-_{Bob}, c)$. Only Bob can open the message because only he possesses the secret key k^-_{Bob}.

In general, the message m which is encrypted with some public key k^+ can be decrypted only with the corresponding secret key k^-. Secret key cannot be calculated or deduced from the public key and vice versa.

In practice, symmetric and asymmetric cryptosystems are usually combined to a *hybrid cryptosystem*. This means that the message m is first encrypted with a symmetric cryptosystem using some shared secret key k_s. The symmetric key is then encrypted using an public key cryptosystem and attached to the encrypted data in some predefined format. Figure 4.7 shows the principle.

[5]One of the design requirements of a public key cryptosystem is, that for a unique public key k^+ there must exist only one unique secret key k^- to ensure that an eavesdropper cannot generate random keys and find another secret key k'^- which would also decrypt messages encrypted with k^+.

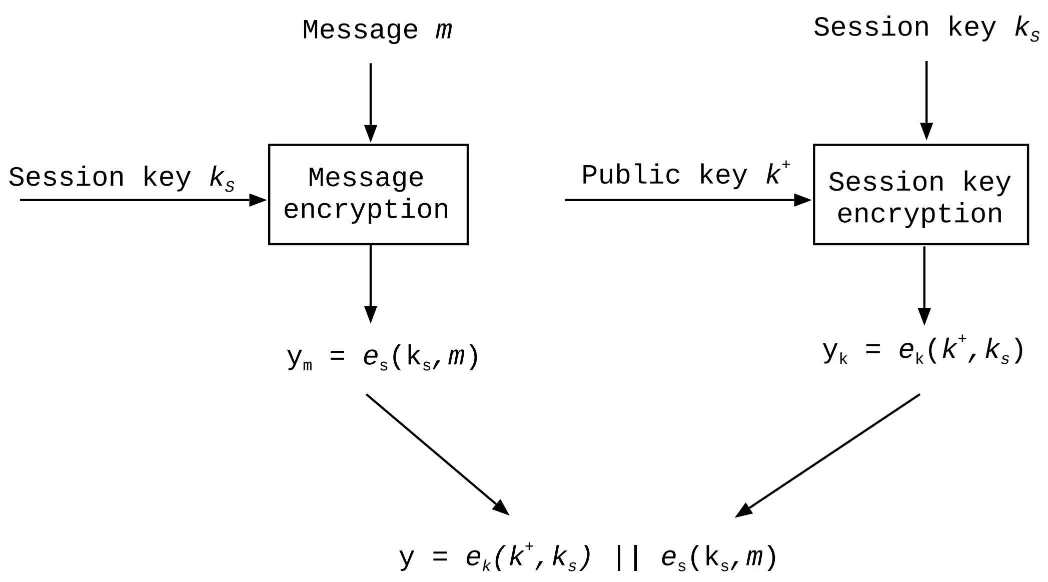


Figure 4.7: A Hybrid cryptosystem

The biggest motivation for the hybrid cryptosystem is efficiency. Symmetric encryption is best for encrypting data because it is orders of magnitudes faster than public key encryption [70]. Further, symmetric encryption cannot be attacked by chosen ciphertext attack like public key systems. However, the symmetric key must also be transferred securely somehow. This problem can be solved by public key cryptosystem as described above.

4.4.2 Digital signatures

For a long time, handwritten signatures are used to prove authenticity and authorship of a document. A signature is also an expression of commitment to some agreement. In *Applied Cryptography*, Bruce Schneier has listed several characteristics of a handwritten signatures [70]: The signature is *authentic*, *unforgeable*, *not reusable*, the *signed document is unalterable* and the signature *cannot be repudiated*. None of these is absolutely true but we can cope with the uncertainty because forging signatures is difficult and the risk to get caught is quite high.

In digital world, the situation is quite different. Bits can be copied and duplicated easily which makes duplication of the signature trivial. Secondly, handwritten documents which are signed cannot be altered afterwards without high risk of detection of modification. This does not apply to digital world either. A mere signature can be easily cut and pasted from one document to another without detection. Therefore the digital signature must be tightly attached to the signed data than the traditional handwritten one.

Public key cryptography can be applied to make digital signatures. The principle is as follows (Figure 4.8): Instead of using the public key to encryption, the private part is used. Since public key cryptography is computationally quite expensive, a hash of the document is signed instead of the whole document. The signed hash and the document are concatenated and encrypted and then sent to the receiver.

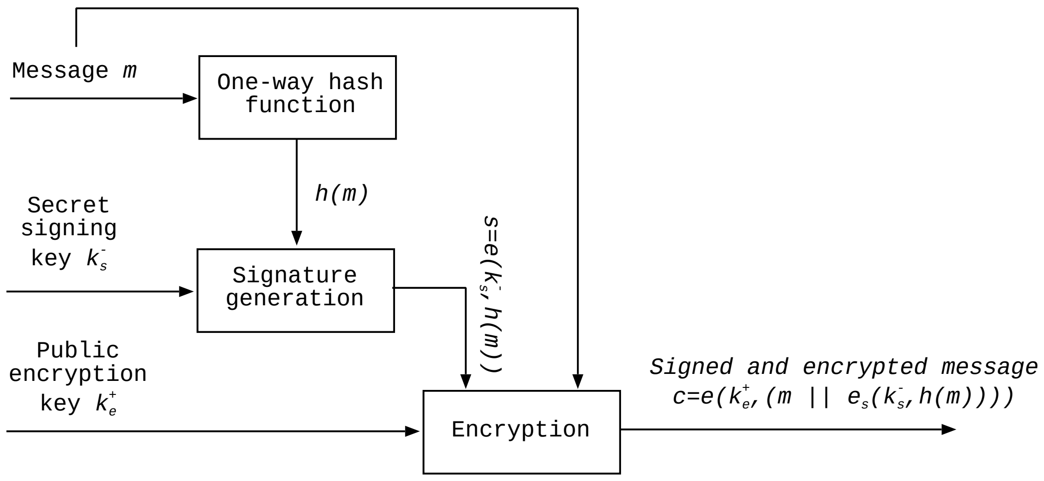

Message *m*
One-way hash function
h(m)
Secret signing key k_s^-
Signature generation
$s=e(k_s^-, h(m))$
Public encryption key k_e^+
Encryption
Signed and encrypted message
$c=e(k_e^+, (m \,||\, e_s(k_s^-, h(m))))$

Figure 4.8: Generation of a digital signature

The order of signing and encryption is crucial. For example, there is an attack against system where encryption is done before signing [70], if RSA is used. Let us consider a situation where Alice want to send a message m to Bob. First, Alice encrypts the message with Bob's public key k^+_{Bob} after which she has $c = e(k^+_{Bob}, m) \mod n_{Bob}$. Then she signs it with her private key and obtains $s = e(k^-_{Alice}, c) \mod n_{Alice}$. (Here n_{Bob} and n_{Alice} are the RSA moduli of Bob and Alice, respectively). Now, Bob who knows the factorization of n_{Bob} can calculate discrete logarithms with respect to his modulus. All he has to do is to find x such that $m'^x = m \mod n_{Bob}$. If he now publishes xk^+_{Bob} as his new public key and still has n_{Bob} as his RSA modulus, he can claim that Alice actually send message m' encrypted with the new key, instead of m. The attack is not limited to RSA only, it works with ElGamal too provided that Bob can choose his own modulus freely [2]. Therefore, the order is to first to make the signature and encrypt after signing. After all, nobody signs anything by hand she cannot read, either. At least she *should* not do so!

4.4.3 Hash functions

Cryptosystems discussed above do not provide protection of data integrity against attackers. If the data being transmitted is altered by a malicious outsider, we cannot detect it. If the data is encrypted and is altered then we are most likely to have some senseless gibberish when decrypting the data.

A hash function is a computationally efficient function which essentially maps arbitrarily long binary strings to specified length string called *a hash value* - some times also called *a message digest*. The second principal property is the ease of computation. Given a preimage m it is efficient to compute the hash value $h(m)$ but it is computationally infeasible to do the inverse mapping[6], i.e. to get m when $h(m)$ is known. For example, SHA-1 (Secure Hash Algorithm - revised) maps a bit string to a 160 bit hash [48].

The idea of hash functions is that even the most smallest change in the input string causes the hash value change radically so that any alteration of the input string is detected. However, since the domain of a hash function is arbitrary large and the codomain is limited, it implies that collisions, i.e. the two distinct images m_1 and m_2 would map into a same hash value h, may occur. This must be kept in mind when hash functions are designed. This property is called *collision resistance*, some literature also calls it *strong collision resistance*. Basically it means that it is computationally infeasible to find any two distinct messages m and m' such that $h(m) = h(m')$.

4.4.4 Message authentication codes

Message authentication codes (MACs) are hash values which are calculated by adding some shared secret part like a symmetric encryption key to the input of the hash function. Hash functions which take a key as a second parameter are called *keyed* hash functions and the output a *MAC-value* or *MAC*. Looking back to the Figure 4.8 the situation is similar, but in addition some secret parameter is fed to the one-way hash function h with the original message m. The requirements of a keyed hash functions are similar to the plain hash functions with two additional claims. First, given zero or more plain text-MAC pairs $(x_i, h(k, x_i))$ it must be computationally infeasible to generate any new text-MAC pair $(x, h(k, x))$ for any $x \neq x_i$ [48]. At a first glance, this does not mean the secrecy of the MAC algorithm but instead the secret key must be known in order to be able to generate new MACs. Secondly, under similar assumptions as above it must be computationally infeasible to recover k by using text-MAC pairs.

The difference of MACs and digital signatures are that the original message can usually be recovered from a digital signature as we presented them in Section 4.4.2 whereas given only a MAC the original message cannot. If compared to hashes, the added value of MACs are that only the party knowing the secret key can calculate the MAC. This is useful feature in virus protection for example. Alice can calculate MACs for her important files. If she used only hashes, the virus could infect the files and after planting itself into files it could calculate new hash values and Alice would notice nothing if she did not keep copies of hash values in some

[6]There exists no function that is proven to be one-way without any assumptions [48]. All applications are therefore more like “candidates” to one-way functions. Therefore, mathematicians consider it possible, thought very unlikely, that truly one-way functions does not exist.

secure place. By using MACs, virus will be detected since it cannot alter MACs because it does not know the secret key.

4.4.5 Tamper-resistant tokens

The concept of authorization certificates implies need to store private keys on some reliable and secure way so that the user can feel relatively safe that the keys cannot get in the hands of misusers. A smart card is one solution for a storage device. Smart cards have memory and processor of their own which are protected from tampering from outside.

However, the computing and storage capabilities of a smart card are rather restricted since the small size dictated by the physical size of such a card. The ideal situation would be that the card itself would generate the keypairs and would only give the public part outside to be listed in some directory service. Traditionally, RSA has been the de-facto algorithm for both public key encryption and digital signatures. The problem is that the RSA key generation is computationally quite an expensive operation. The two large secret random numbers p and q that RSA is based need not only be random but also primes [48] at least with very high probability. It is the testing for primarity which is expensive. Therefore, public key cryptosystems based on elliptic curves are more attractive solution to be used with smart cards since elliptic curve cryptosystems (ECC) need only a random number which need not to be prime.

Tommi Elo studied an application of elliptic curve digital signature algorithm (ECDSA) on a smart card environment [20], namely on a JavaCard[7], in his master's thesis. His conclusions were that performance of currently available smart cards cannot meet the feasible requirements for speed if implementations are done in software. Although the implementation is possible, the response time of the JavaCard would be far too big for the needs of any commercial product. Even the actual availability of the cards which some manufacturers advertised and told they had released was highly questionable.

[7]JavaCard is a registered trademark of Sun Microsystems, Inc.

Chapter 5

Distributed Access Control Management System

5.1 Previous work

Theory of access control has been studied for a long time. Sandhu et al have done extensive research on AC and AC models based on roles [68], and lattices [67]. However, these studies have been based on the assumption that first the subject is authenticated, i.e. the identity of the subject is verified. Secondly, the rights of the user are inspected, in this case what kind of access the subject can be allowed. Sandhu states that authentication is one building blocks of information security [66]. Mostly Sandhu's and his research fellows' work have been based on identity-based AC but they have noted that the role based AC is also important and in many cases more interesting than identity based.

Blaze and Feigenbaum have taken a different approach to the problem. As we have described earlier, they have studied the problem from a more general point of view, i.e the *trust-management approach* [9–11]. They have found out that today, identity carries insufficient information in order to be a basis for authorization decision in a distributed system. A better solution would be to base the authorization decisions on credentials which a public key presents and which are compared against a local security policy [8, 24, 25].

Ellison and the SPKI working group have worked on the same problem as Blaze and Feigenbaum. SPKI is a step towards an architecture of authorization, not identification. SPKI forms part of the foundation on which TeSSA architecture is built. Nikander has discovered the same insufficiency of the identity as the basis of authorization as Blaze and Feigenbaum have done. TeSSA-architecture [54] has been developed from the perspective of authorizing actions.

In TeSSA Research project, Heikkilä and Laukka have implemented a distributed banking system using TeSSA architecture [32]. We are applying the TeSSA architecture to implement a physical access control system based on SPKI authorization certificates.

5.2 On the Basic Problems of Access Control

Basically, the function of physical access control is very simple: To allow authorized personnel to enter the building and forbid unauthorized entry. Of course, the real problem is much more fine-grained and cannot be simplified this drastically. The mere definition of ’authorized’ access is not unambiguous. Let us assume that the building is an office and the people who work there do some ’general’ office work. Which one of the persons in the following two cases has greater authority to enter the building, the ordinary guy who works there, or an errand boy who delivers some computer hardware that was recently ordered from a local store? In both cases the person has an equal justified need to access the building.

In the following, we list seven common problem areas of physical access control systems.

Key distribution. Ordinary workers are quite an easy case. The access control manager gives the appropriate keys to the persons when they start working in the company. But, how to distribute keys to the authorized users who might need them only temporarily, like our courier boy in the above scenario, is a bigger problem.

Returning the keys. Usually the old keys are a larger problem than distributing the keys. If old workers do not remember to return the keys when they leave, they have a possibility to come back with malicious purposes. Locks can be reconfigured so that old keys will not be valid but that would be enormous task every time.

Key duplication. How to ensure that unauthorized parties cannot duplicate valid keys. Most advanced physical keys have usually patents which protect both the keys and the lock industry so that only the original manufacturer can make more duplicates for the customers.

Key borrowing. How to prevent the authorized users from borrowing their keys to unauthorized persons?

User information management. How to administer the user information, i.e. personal information and access rights effectively.

Recognizing information. If we are using physical keys the situation is quite simple. Either the key opens the lock or does not. In digital world, the problem is much more complicated because physical keys cannot straightforwardly be turned on digital form.

Different roles. Very often people are acting on some role when they do their everyday duties. A computer administrator have access to a computer room of the computer center which is otherwise strictly guarded.

The above list depicts some of the common problems which are usually noticed but in practice very often misused. For example, key borrowing is very common practice despite the fact that a corporate security policy most probably denies it.

Based on the observations above, we describe more detailed requirements and criteria for a better solution in Section 5.6.

5.2.1 Why Do Distinguished Names Fail?

In centralized systems, where access control have generally been implemented with usernames and passwords, authorization is usually divided into two distinct subproblems. First is the identity, i.e. “who made the request?”. After the first question is answered, the second one asks “is she authorized to take the action?”

Figure 5.1 depicts the general problem. If a distinguished name of a person is used as the basis of the identification, we are using the binding (a) i.e. Person $\rightarrow$ DN to link some real person to a subject in cyberspace. Now the first question “who made the request?” is answered. Now we need to know the answer to the question “is the requester authorized to take the action?”. This can be checked from an access control list (ACL). For example, a UNIX filesystem keeps information of the permissions of every file in the system which consists of three flags, the owner, the group, and the other which controls the access to the file. This is the binding 2, DN $\rightarrow$ authority in the Figure 5.1.

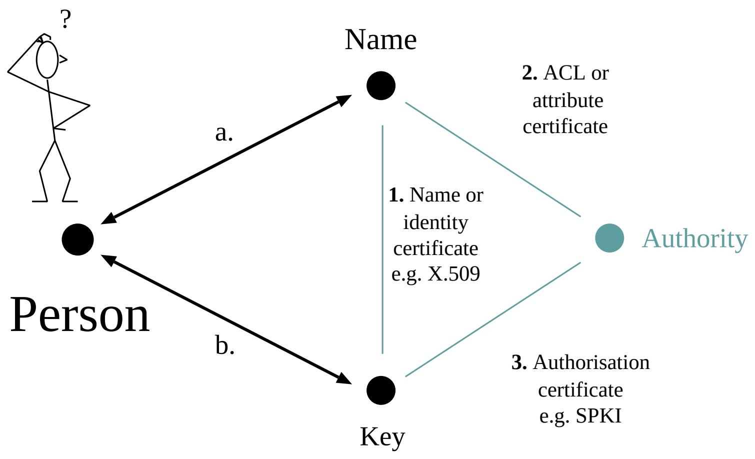

?
Name
a.
2. ACL or attribute certificate
1. Name or identity certificate e.g. X.509
Authority
Person
b.
3. Authorisation certificate e.g. SPKI
Key

Figure 5.1: The authorization problem

In Finland, the estimated number of users of Internet by Nua Ltd. is about 2.27 million people [57], which is roughly the number of working aged people in Finland. If we look the statistical figures of names, the most common familyname is Virtanen. In March 2000, there are 24,998 people having the surname Virtanen in Finland according to the Population Register Center [78]. The most common male first name in Finland is Matti [3]. Combining these, we can make the assumption that quite probably there exists at least two distinct persons with the name ’Matti Virtanen’. Therefore, a mere name cannot be used either as a piece of identifying or authorizing information in either a nationwide network or a global network as the Internet.

Even more complicated is the usage of name-certificates which bind a distinguished name to a public key (binding 1). As the public key is the reasonable representer of a person in the networked world, we also need some authorization given to the public key to make some actions. This final binding is done by $\mathsf{key} \rightarrow \mathsf{authority}$ mapping in Figure 5.1. We had to do three steps in order to get answer to the basic question “Is the person authorized to take the action?”

The Figure 5.1 suggests that the easiest way to get the answer is to take only two necessary steps from a person to the authority. First is to bind a name to a public key and then bind the public key to some authority. Name is not needed at all. This is also the way how the system works in the “real world”. As we showed in Section 5.3.1, traditionally a physical key is given to a person who uses the key to access a locked facility. No identification is done in the actual access control situation when the person opens the door.

5.3 Access Control Architecture

5.3.1 A Centralized Solution

The principal guidelines of an access control management system are depicted in Figure 5.2 In phase one the person acquiring the access to some facilities goes to meet the access control manager (ACM) to have a key[1] to the facility. ACM checks the identity of the user and gives an appropriate key to the user. Usually the user must sign for the key so that the ACM can keep track of the keys which are given to the people. This information is stored to a secured database. The important thing to notice here is that the identity of the user is checked here. If everything is in order then the key is given. If it turns out to be so that the user is some imposter then the fraud is detected here (or at least it should be!).

Now the user has a key so she can enter the facility. Depending the implementation of the access control system, there are two general alternatives.

First is the case of a plain, conventional lock and a physical key. A physical key is most usually very difficult to forge or even manufacture. Moreover, the most advanced keys are patented so that nobody else than the patent owner can manufacture those keys. If the legal user wants a duplicate key, it must be ordered from the manufacturer of the lock who will make duplicates only to the original purchaser of the specified lock. The conditions which the user of the key commits herself, among others, when she receives the key is not to give to key any other person. These procedures and conventions ensures that the ACM can be quite sure that only authorized personnel can enter the facility being protected. There is, in most cases, no reception or guard in the doors who would check the key owner’s identities. Only facilities which require very high security might take this action to ensure high security. A national central bank would be an example which requires such a high security level.

The second alternative is the electronic key, a magnetic key for example. The most usual implementations are such that every key has some unique identifying

[1]Here we the term ’key’ to mean a very *general* key, it might be a traditional physical key as well as a magnetic key or a smart card based one.

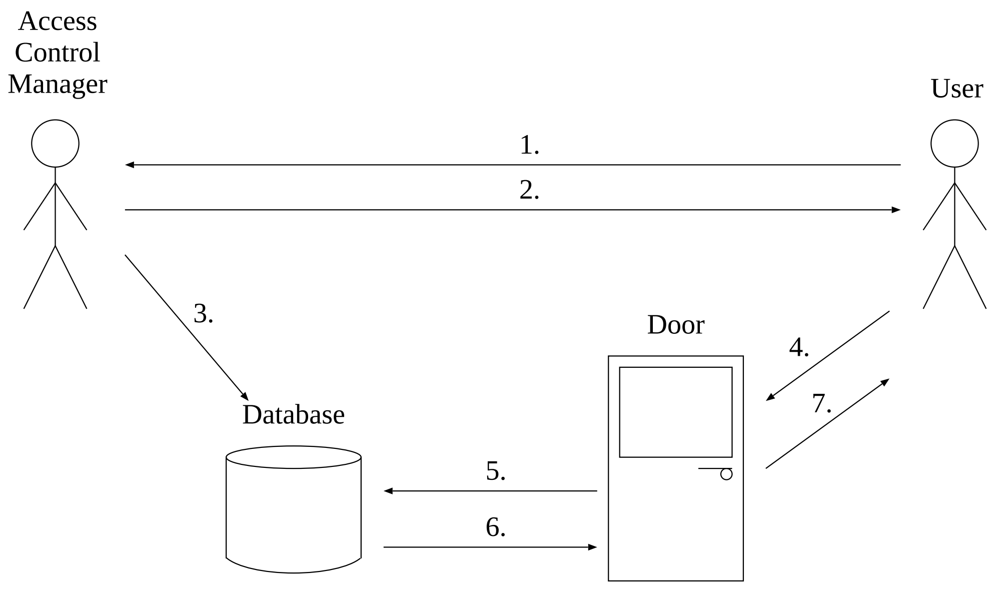


Figure 5.2: A traditional access control management system

number which individualizes that particular key. The door does not know which keys are valid ones so the door must make an inquiry to some central database asking whether or not to grant access. This is the step 5 depicted in Figure 5.2. If the key which was presented is valid then the database system tells door to grant access i.e. to open (steps 6 and 7). The trust relationships are similar to the previous situation with the physical key.

However, there are a few drawbacks here. Firstly, the system requires a central database to store the valid key-IDs (if electronic keys are used). If the connection to the database is broken or the database it self is down, the whole system is inoperative. The other alternative is that each lock would knew which are valid keys and which are not. This is not a realistic assumption since it would require to somehow update the information of all the locks every time a new key is introduced or an old one is revoked. If the environment in which the access control system is used grows large enough the task of updating each lock becomes too big. In case of physical keys, they must be hard to forge, and the keys must be collected back when person does not need them anymore.

5.4 A Distributed Access Control Management System

Figure 5.3 shows the general architecture of our certificate-based, distributed solution for an access control management system (ACMS). Conceptually the setting is as follows: to be a truly distributed system, each door is the beginning of the authorization. A door can issue a certificate in which the door gives permission to grant access rights to that particular door. Every door automatically issues such a certificate to the access control manager (ACM) of the system. This certificate basically gives the ACM total authority over the door. Also, the permission to further delegate the access control is granted.

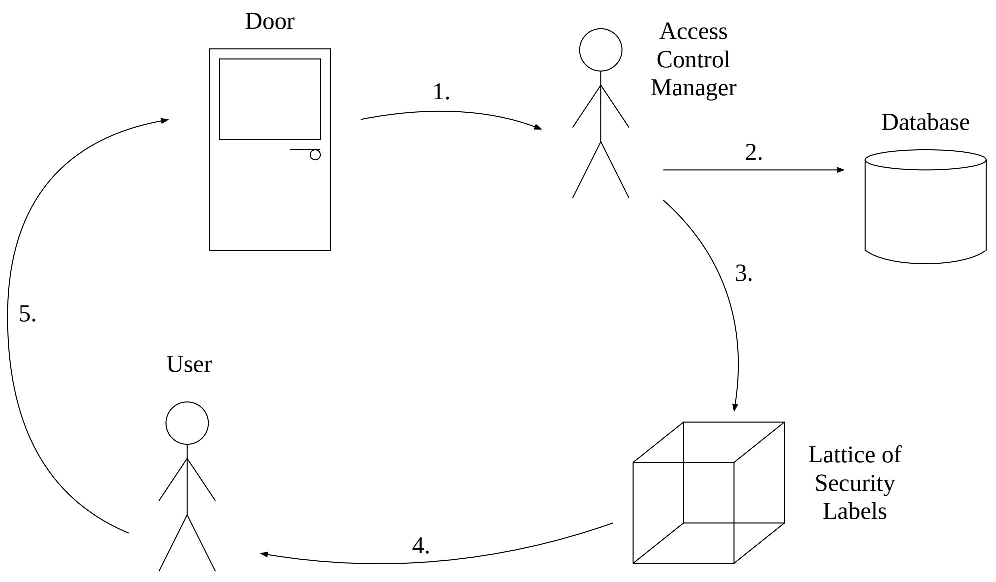


Figure 5.3: Distributed access control system

The AC management is distributed also. An organization usually has some person who is responsible for the general security of the organization. However, since the organization might be quite big, it is not wise to define the management of the system to be constrained to one physical person. We will therefore assume that there is one organization level security officer who is responsible for the organization-wide security. However, she cannot really do all the job by herself. Therefore she has divided the access control management to smaller parts according to the organization structure. Every organizational unit has a access control manager of their own, who controls the smaller organizational unit. Her job might be further split to smaller parts. These split ups can continue as many times as needed. The access control management part of our system system is actually a tree which has the chief security officer as a root node and leaf nodes are security managers of different smallest units of the organization. The Figure 5.4 shows the general model of such a tree.

The organization manager who has permission to give access to any building or facility in the organization issues certificates to every department level security manager which grants permission to give access within the particular department. Depending the organization size, the tree can have as many levels as necessary. For example, in the leaf nodes are the managers of the smallest units which can be offices or laboratories. These office level managers can define the local secu-

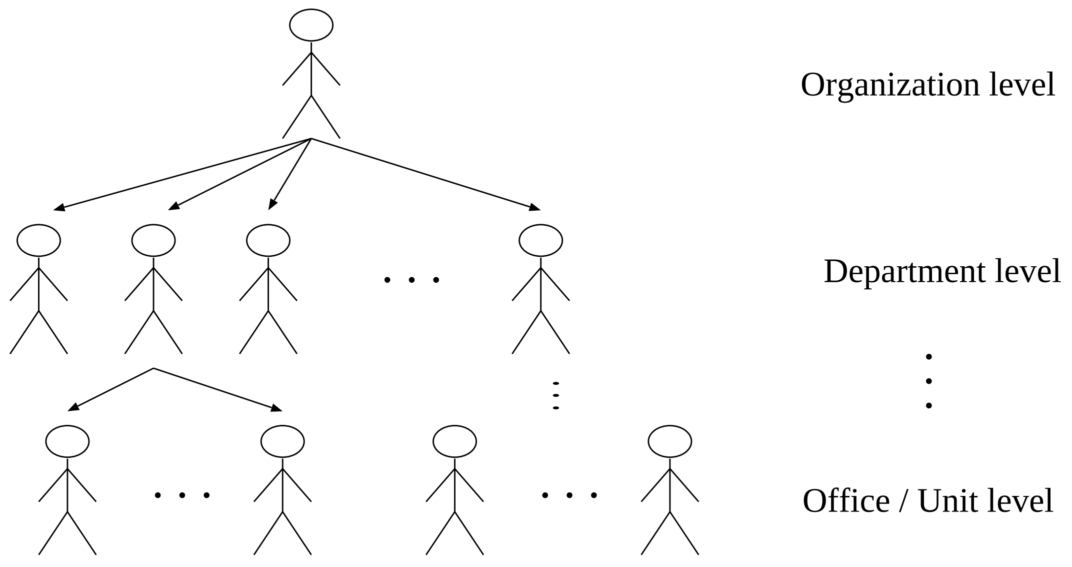


Figure 5.4: Access control management tree

rity policy of the particular office which defines the rules and access permissions in that office. Different offices can have different needs for a security policy and therefore it is the manager of that particular office who knows the local needs the best. Usually there are organization-wide general guidelines of security policies but the more fine-grained policy is created in every office according to the guidelines of the general policy. Arranged as described above, the access permissions which are delegated onwards on each step form a sequence $p_1 > p_2 > \ldots > p_n$ where p_1 is the organization-wide access permission on the root of the tree in Figure 5.4 and p_n the office level permission on some leaf of the tree.

The manager of each office writes a certificate to the people of that office with appropriate access rights. In the office level, the security policy and the access rights can be modeled with a lattice of security labels as we described in Section 2.10.6. Amoroso [1] and Gollmann [27] give some examples of security labels with categories grouped by rather broad level, like departments in an organization. However, because lattices are rather difficult to visualize by Hasse diagrams if the lattice becomes complex and large [1]. Therefore it is better to introduce security labels only as the final solution in a office level, not any higher. However, nothing prevents using security labels also by the non-leaf nodes of the AC management tree. This might be useful in a following scenario. Let us assume that more than one offices or organization units are located on a same building which quite often is the case. It is more convenient that the AC manager responsible for the building-wide security issues the certificates which grant access to the building instead of the office level AC managers.

The access control manager in the leaf node issues a certificate to the end-user who actually uses the certificate and passes through the doors. The leaf ACM has defined appropriate local security policy and classifications to each door under her administration. Following the policy, she can now issue a certificate with an appropriate clearance to the end-users. The end-user certificate expresses the clearance level explicitly - for example, a computer system administrator could have clearance

$$(\mathsf{restricted}, \{\mathsf{computer} - \mathsf{admin}, \mathsf{staff}\})$$

Now the user can access the facility with the certificate which she has. Depending of the solution of the access control system, the certificate could be stored in a smart card or a trusted PDA. Maybe even on a cellular phone which can communicate with the door via an IR link.

5.5 On Certificate-Chains of the Access Control Architecture

To further simplify and make the concept clearer, we will describe the certificates and certificate chains in more detail now.

First, the door has issued a certificate to the ACM on top of the access control management hierarchy of the organization in question. This certificate grants the ACM permission to totally control and grant access permissions of the door. This is represented as certificate Cert.1

$$(D, ACM_0, \mathsf{true}, ''\mathrm{ManageAccess}'') \tag{Cert.1}$$

Since this permission is valid forever, there is no validity field $\mathcal{V}$ in the certificate [53]. Because the SPKI certificate specification does not give any description of how the authorization information should be presented [19], we can choose the representation form freely. Here we want to emphasize the concept itself so we decided to encode the authorization information in the field $\mathcal{A}$ in a human readable form.

Depending on the structure of the organization, the structure of the ACM tree described in the previous section may vary a lot. However, in generally there is some path $n_0, n_1, \ldots, n_l$ on the tree from the root node n_0 to some leaf node n_l. The authorization from the head ACM to the leaf ACM is represented by certificates Cert.2,...,Cert.4

$$(ACM_0, ACM_1, \mathsf{true}, \text{“ManageAccessOnDivA}'') \tag{Cert.2}$$

$$(ACM_1, ACM_2, \mathsf{true}, \text{“ManageAccessOnDeptB}'') \tag{Cert.3}$$

$$\vdots$$

$$(ACM_{l-1}, ACM_l, \mathsf{true}, \text{“ManageAccessOnOfficeZ}'') \tag{Cert.4}$$

The above chain shows how authority information flows from upper to lower levels. First, from the chief ACM ACM_0 on the root to the ACM_1 of some division of the organization, and after some steps to the office level on the hierarchy. On every step the authorization becomes a bit narrower than previous level as intended.

Here the ACM_l is responsible for describing a security policy on office level. Depending of the lattice structure she has defined, she issues a certificate Cert.5

$$(ACM_l, u, \mathsf{false}, (\mathsf{restricted}, \{\mathsf{computer} - \mathsf{admin}, \mathrm{staff}\})) \qquad \text{(Cert.5)}$$

Here the authorization field $\mathcal{A}$, namely $(\mathsf{restricted}, \{\mathsf{computer} - \mathsf{admin}, \mathrm{staff}\})$ is the clearance of the user u. The user u is not allowed to further delegate the access permissions. Actually, this depends on the local security policy. It might be allowed for end-user to further delegate the access permission. Further delegation might be restricted to only a one step. Since delegation field of the SPKI certificate is binary in nature, this must be specified in the authorization field with the clearance.

When the user u access the door she presents the certificate Cert.5 to the door D. Since the access of the particular door happens quite frequently, it is reasonable that the door D stores the certificate Cert.1 locally so it can be accessed fast. Once the certificate chain of distributed ACM is complete from the first certificate Cert.2 to the leaf certificate Cert.4, it can be reduced to a single certificate Cert.6

$$(ACM_0, ACM_l, \mathsf{true}, \text{“ManageAccessOnOfficeZ''}) \qquad \text{(Cert.6)}$$

The door D can now check that the chain of certificates Cert.1, Cert.6, and Cert.5 is valid. Since door D is the source of authorization and it knows the classification of itself, it can conclude that the the access may be granted provided that the clearance of the user u dominates the classification of the door d. Otherwise the access is denied.

5.6 Design Criteria

In previous section we described a centralized access control system from which we pointed out several drawbacks. We will now describe and state criteria for a better solution.

The new access control system should be based on **authorization** to go through doors. Physical key is the authorizing token which have been used since the locks were invented some 4000 years ago [13]. There has been no constant identification after the key has given to a person.

Access control system should be truly **distributed** without a need for a centralized database. Access control decision should be possible to make without any queries to any databases or at least any such database should be distributed. Further, the management of access control rights should be distributed. Various recent studies have shown [24, 46, 54] that large distributed systems has various demands for security which differs in different parts of the system.

Key-management of the system should be flexible and scalable and yet secure. By flexible we mean a system where old keys, for example, do not comprise a problem like the situation is with physical keys if administered poorly. It should be possible to describe whether the key holder may give or borrow the key to another person temporarily.

Unnecessary log-generation should be avoided. **Privacy** of the legitimate key holders should be respected. There cannot be any arguable reason to monitor the movements of legitimate users. Only those who are trespassing or otherwise making an unauthorized access should be tracked and detected. Working time calculation is totally another problem and is out of the scope of this work.

Access control checking should be as **efficient** as possible. Using new technology for access control should simplify ordinary tasks and enhance the security. If opening the door takes too long time, people only get irritated. An optimal performance would be faster than one second for door to make the access control decision. Lower than a second response is like a blink of an eye for human beings [20].

Chapter 6

Implementation

Based on our conceptual design introduced in the previous chapter we will now describe our implementation in more detail.

6.1 Architectural Elements

6.1.1 Unified Modeling Language

As the name says, Unified Modeling Language (UML) is a modeling tool for software development process. The UML was standardized by Object Management Group (OMG) and the current version is 1.3 [81]. UML is a *modeling language*, not a *method* of design. Usually, methods consist both a modeling language and a *process*. The process describes which are the steps in the path from the scratch to a fully functional program.

UML offers many different graphical tools for design task. Fowler has described them quite thoroughly in his book *UML Distilled* [26]. We found use case diagrams, class diagrams, and interaction diagrams most useful.

A *use case diagram* is a tool for identifying different actors of some use case. A use case is a set of scenarios which are tied together by some common user goal [26].

A *class diagram* describes different types of objects and the static relations between them. It is the most important tool of the UML because of the power of expression [74].

An *interaction diagram* is a tool for modeling how objects collaborate and work together [26]. Interaction diagrams are further divided into two subtypes, i.e. *sequence diagrams* and *collaboration diagrams*. A sequence diagram show how objects send messages to each others and how things happen in a sequence. A collaboration diagram show the static connections between objects.

We found the use case diagrams most useful in the beginning of the development phase to distinguish the different parties and actors of the access control management problem. The interaction diagrams were very useful soon after the beginning when we studied how the different pieces of information flows in our system. In the end, the class diagrams were the main tool to describe the objects and classes of the access control management system. An interaction diagram was used to illustrate the steps how the access control decision is done in practice.

We tested several different UML tools during the design phase. After trying ArgoUML[1], Dia[2], and MagicDraw 4.0[3] we found the latter the most useful tool.

6.1.2 Java

The Java programming language has been developed at Sun Microsystems Inc. The goals that the designers of Java programming language set in the beginning were to develop a programming language which is *simple*, *object-oriented*, *distributed*, *robust*, *secure*, *architecture neutral*, *portable*, *interpreted*, *high performance*, *multithreaded*, and *dynamic* [34].

Java is an interpreted programming language. The Java source code is compiled into Java *byte code* which is executed in a *Java Virtual Machine* (JVM). The principle idea is that the Java byte code compilers and JVMs can be implemented on different computer architectures and the actual bytecode is thus architecture independent.

Security of Java code is based on checks done by JVM on run-time. JVM provides a so called sand-box [60] which gives certain basic resources for the byte code being executed. The byte code may quite freely operate inside the sand-box but it needs some special permissions if it wants to access any external resources, like disks or printers, which are outside the sand-box. This is an effort to try to minimize the possibility of pest programs like worms or Trojan horses or viruses to do malicious things.

The syntax of Java resembles closely C and C++ which makes Java easy to learn if either of these programming languages is already familiar. To avoid potential sources of bugs and programming errors, there are no pointers in Java. Further, the memory management is automatic, i.e. the programmer need not to take care of memory allocation for objects being created. Also objects need not to be explicitly deleted because JVM contains a garbage collector which cleans memory for objects which are no longer being used.

The choice to take Java as a programming language for our prototype was quite obvious, since all the programming in TeSSA-environment has been done in Java.

[1]`http://argouml.tigris.org/`

[2]`http://www.lysator.liu.se/~alla/dia/dia.html`

[3]`http://www.magicdraw.com`

6.1.3 Java Database Connectivity

Java Database Connectivity (JDBC) is a Java application programming interface (API) to use any kind of tabular or relational data [36]. The first version of JDBC was released in summer 1996 by Sun Microsystems Ltd. JDBC allows programmers to access databases through a Java API by using Structured Query Language (SQL). Still today, databases are one of the largest application area of computer systems. Therefore, the JDBC was one of the largest improvements for Java environment because also the JDBC API is platform independent so an application accessing database written in Java can be executed in any platform running any database system [35].

JDBC consists of two layers [35]. On top of the stack is the JDBC API which provides classes and methods to execute, query and update database using SQL statements. Second from the top is the JDBC driver manager which receives the SQL statements and communicates with a database vendor-specific driver. JDBC driver manager connects to the actual database and executes the queries and updates and returns the results from the database to the upper layers. The upper layers encapsulate the data which is arranged in tabular form into Java class structures.

Programmers can thus implement applications with Java programming language and access database from the application by using the familiar, standardized SQL statements. The lower level drivers hide all vendor specific features of the database being used so that the programmer can concentrate on programming universal applications which can be executed independently from the hardware platform. To make this possible, all JDBC drivers must comply with at least SQL 92 standard [35].

6.1.4 Semantical Overview

When we started to outline the features of the ACMS we adopted a design method which studied the problem from the user point of view. We tried to divide the problem into smaller sub problems based on different kind of use cases.

As a result of our initial analysis, we identified four different subproblems which led us to a structure depicted in Figure 6.1. Access control management system is divided into four different lower level tasks. These tasks are *User management*, *Local Security Policy Management*, *Trust Delegation Management*, and *Compliance Checking Daemon*. We will describe these tasks in more detail in the following sections.

6.2 User management

A more detailed class diagram of User management part is depicted as a UML class diagram in Figure 6.2. The user management part is responsible for col-

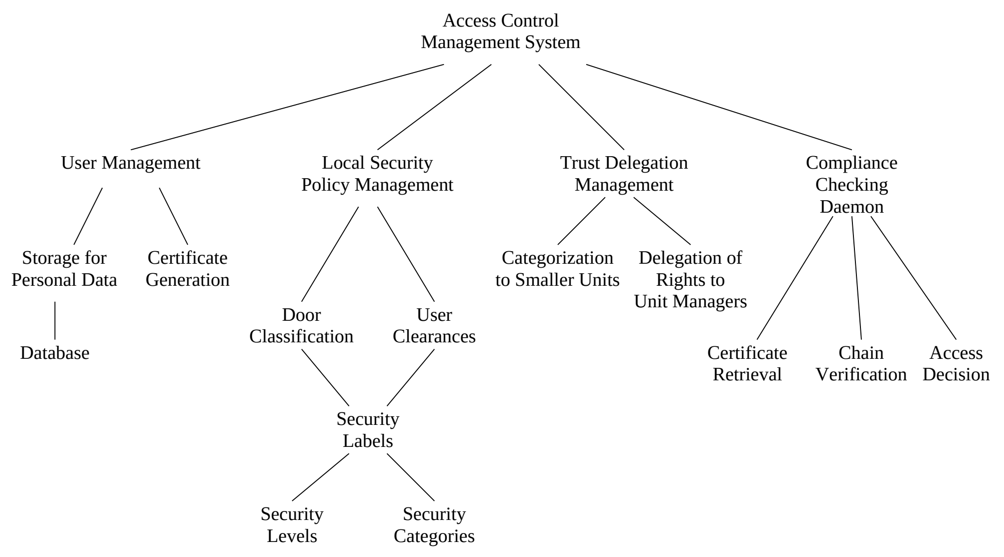

Access Control Management System
User Management
Local Security Policy Management
Trust Delegation Management
Compliance Checking Daemon
Storage for Personal Data
Certificate Generation
Database
Door Classification
User Clearances
Security Labels
Security Levels
Security Categories
Categorization to Smaller Units
Delegation of Rights to Unit Managers
Certificate Retrieval
Chain Verification
Access Decision

Figure 6.1: Semantic architecture overview of the prototype

lecting the necessary user information of the workers of an organization. This information includes name and address, phone numbers and email address and birth date[4]. User information is collected into a `User` class which contains the necessary utility methods for parsing and processing different date and time representation forms, for key and certificate generation, and for representing user public key in a more readable BASE64 encoded form.

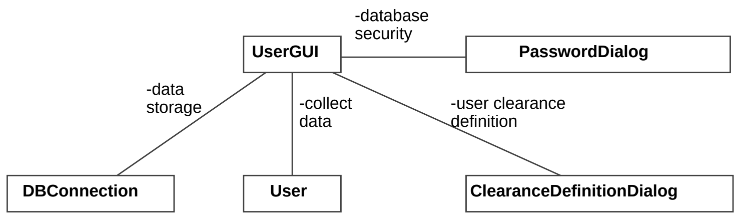
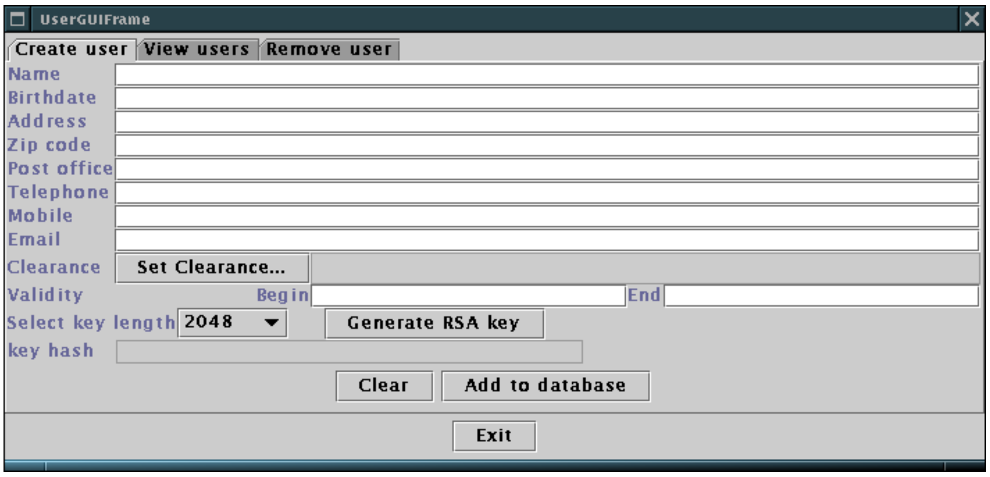


Figure 6.2: A class model of user data management

The graphical user interface (GUI) for the user management is shown in Figure 6.3. The necessary information like the clearance for the current user is defined here among the total validity time of the certificate. The certificate for the user is generated here. After the certificate generation the user data is added to the database. The certificate is also stored in the database since it contains the information of the user access privileges.

Personal data of the users is stored in a secure database. Since our implementation is done on a RedHat 6.2 Linux on a Pentium II 333MHz we chose PostgreSQL as

[4]We want to emphasize that we store only the birth date, *not* the social security number.



Select key length 2048

Figure 6.3: A screen shot of the graphical user interface of user management

our database solution since PostgreSQL is included in the RedHat distribution[5]. PostgreSQL development team has just received the 2000 Linux Journal Editor's Choice Award for Best Database[6]. Therefore we believe that PostgreSQL is one of the best open source databases available to choose from.

The database is protected with usernames and passwords to block out unauthorized access. When the user data is updated to the database, the GUI shows a password dialog which is implemented in the `PasswordDialog` class. `DBConnection` encapsulates all the necessary information and methods to access the underlying database.

6.3 Trust Management and Delegation

Trust management and delegation handles the distribution of the AC management into different smaller organizational units as we described in Section 5.4. Each organizational unit is represented as an `OrganizationUnit` which can contain more sub units which in general form a tree structure. Every unit has one or more trust managers - more than one is needed since it is wise to have a vice trust manager if the primary manager is sick or on a work trip or otherwise prevented from doing her duties. However, one trust manager is responsible for only one unit at a time. This ensures the true distribution of AC management.

Trust manager defines the local security policy which is encapsulated in `LocalPolicy` class. Local security policy and `LocalPolicy` class is described in Section 6.4 in more detail.

[5]`http://www.redhat.com`

[6]For more information on PostgreSQL databases, see the homepage `http://postgresql.rmplc.co.uk/`.

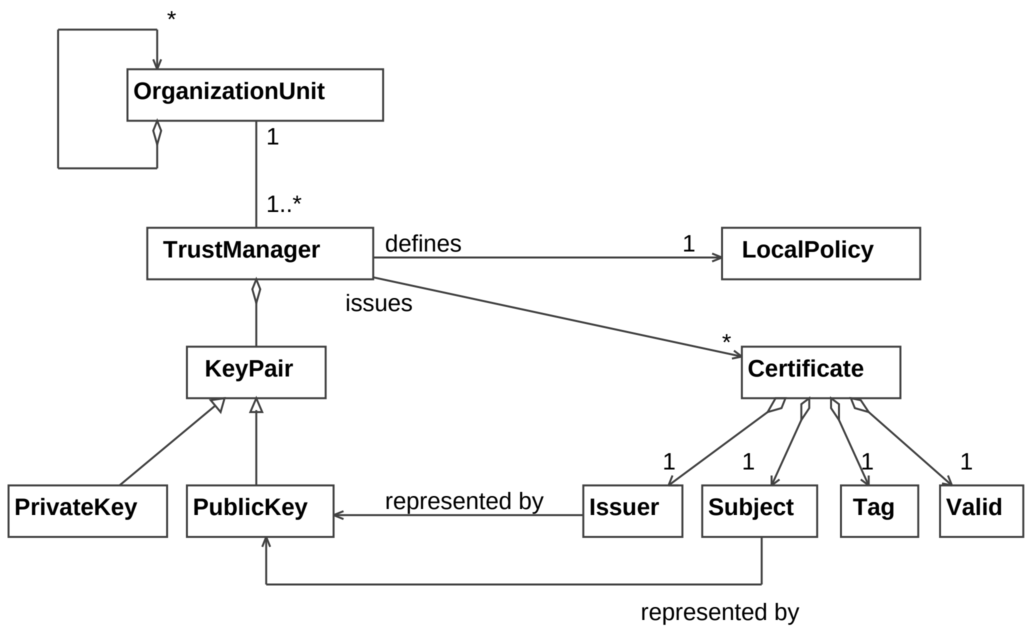


Figure 6.4: A class model of trust management and delegation in an organization

Every trust manager has a certificate which contains a public key. Trust manager issues the end-user certificates for the users with appropriate classifications. The class diagram in Figure 6.4 shows the class structure. In the certificate, we included only the classes which are interesting from our point of view. Also, the certificate has a couple of more subclasses, for example `Comment`. An exhaustive list of sub classes can be found from [44, 60]. In the certificate, `TrustManager` is represented by the issuer which essentially is a `PublicKey`. The `Subject` is the public key of the end-user. `Tag` and `Valid` classes are described in more detail in Section 6.4.

6.4 Local Security Policy Management

Structure of local policy management is shown in class diagram in Figure 6.5. A local policy which is defined by a local security manager consists a set of users and door groups. Door groups contains one or more doors. It is very likely that there is several groups of doors with equal classifications. By grouping them together as door groups the management becomes easier.

End-user may have one or more certificates. This is necessary in the situation which we already outlined conceptually in Section 5.4. Several offices or small departments may be in one large building. It is practical that the access permission to the building front doors is defined in another certificate for several reasons. First, the person might work in several organization unit at the same time. This

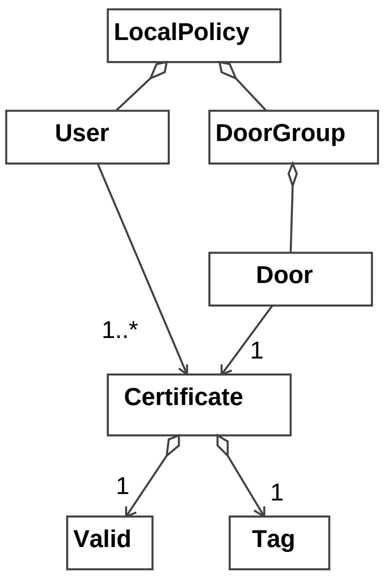


Figure 6.5: A class model of local security policy management

is possible at least in a university environment. These units might have different restrictions for access and therefore the access to the building cannot be defined on based only one organization units needs.

Doors, however, have only one certificate, i.e. the one in which the door have delegated all the access control to the ACM - or trust manager - on top of the organization.

`Valid` field describes the validity period of the certificate and thus the validity of the access permission. Validity of an SPKI certificate is defined by two dates d_1 and d_2. Between these dates, i.e. on the closed interval $[d_1, d_2]$, the certificate is valid.

However, a such simple validity period is not necessarily enough on an access control situation. It might be required that the access permission is valid only on working days, i.e. starting on Monday and ending on Friday. Further, it might be necessary to state some daily time limits also. Access permission might be valid only on office hours starting at 8 am and ends at 4 pm.

To allow a more fine-grained definition and restriction of access permission, we decided to include an additional limit entry in the `Tag`-field. We extended the basic `Tag` definition of the SPKI certificate by defining a special `DoorTag` class which inherits the basic properties and methods of SPKI `Tag`. A `DoorTag` is constructed from two different parts. Depending on the owner of the certificate the tag contains either a `Classification` or a `Clearance` for `Door` and `User`, respectively. Both the `Classification` and `Clearance` are basically `Se-curityLabel`s but we decided to follow the traditional naming convention [1]

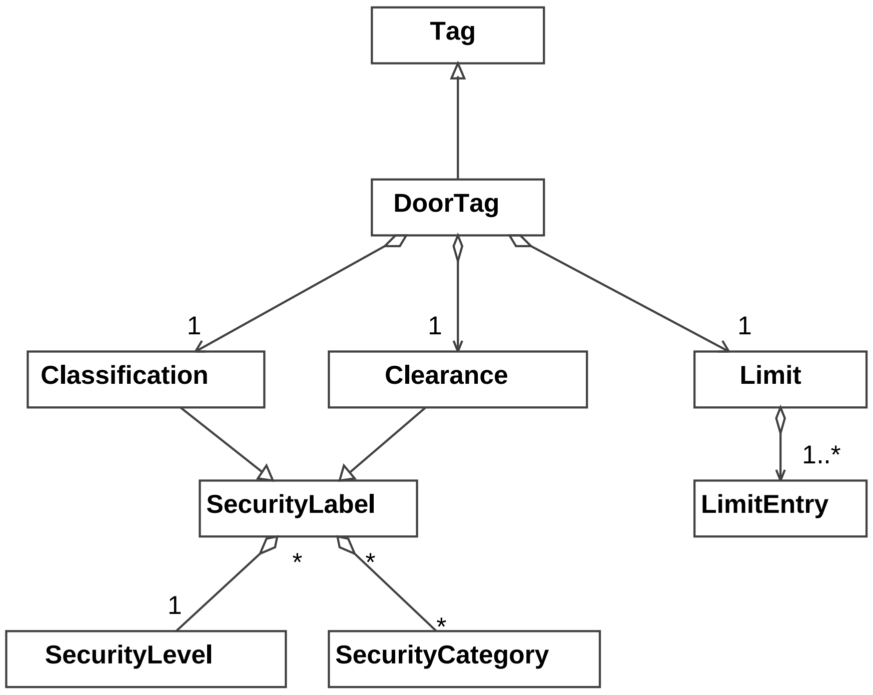


Figure 6.6: Our extensions to the tag field to express building AC information

which has been used for some time in the access control field. A `SecurityLabel` contains a `SecurityLevel` and a set of `SecurityCategory`s as we described earlier in Section 2.10.2.

Time constraints of the access permissions are represented by a `Limit` class which can contain one or more `LimitEntry`s. This way we can express fine-grained time constraints in a very detailed level.

6.5 Access Control Decision

When a user comes to a door and presents her certificate, the door must make an access control decision whether to let the trespasser in or not. Each door runs and instance of an `AccessControlDecisionDaemon` which takes the certificate and evaluates it. The class structure is depicted in Figure 6.7. The `SPKIServices` is an existing utility class to search certificate chains and evaluate tags.

When the door receives the certificate, it first checks whether the certificate is valid. The required information is easily checked from the certificate `Valid` object. In a positive case, the door collects the necessary certificates by applying the `searchForChain()` method of `SPKIServices`. If the chain can be constructed, the door then applies the `checkPermission()` method of the

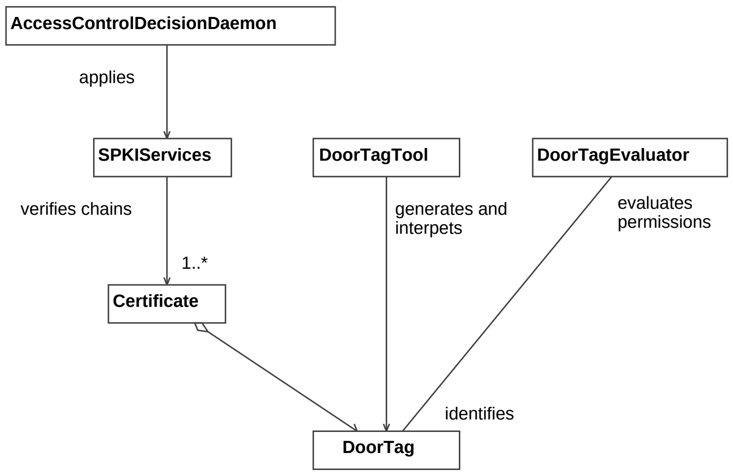


Figure 6.7: A class model of access control decision checker

`SPKIServices` with the previous chain and the `DoorTag` object from the user certificate as parameters for the method.

Since the SPKI documentation does not specify any pre-defined structure for the tag field, the application programmer must first introduce the tag structure and then implement an application specific evaluator for the tags. We have implemented a `DoorTagTool` which generates and interprets `DoorTag` objects. Secondly, since tag has no standard form, the `DoorTag` object must know and be able to tell the object which knows how to evaluate these specific tags. We have implemented also an evaluator for door tags. `DoorTagEvaluator` implements the general `PermissionEvaluator` interface [60].

`DoorTagTool` constructs and interprets access permissions as follows. Consider an example from a university environment. The door gives the first certificate to the ACM as described in Figure 5.4 in Section 5.4. The tag in this certificate includes the permission 'hut'. Let us say that the root ACM then issues a certificate for the ACM responsible for the Department of Computer science and the Computer Science building. The tag in this second certificate includes a permission 'hut.cs-dept'. The latter ACM further delegates the authority to control TML laboratory's access control to the laboratory engineer. The certificate which the laboratory engineer receives has the permission 'hut.cs-dept.tml-lab'. Finally, the end-user receives a certificate with a tag, say, 'hut.cs-dept.tml-lab :(restricted,{computer-support,admin}):0800-2000'. This tag gives the user permission to access doors belonging to door group 2 in TML laboratory between 8 am and 8 pm. In general, the end-user tag is constructed of three parts. First is the definition of the building or other organization unit which depends heavily on the organization structure. In the second is the classification of the user. Third and final part is the time limitation of the access permission in the unit level.

Constructed this way, the first parts of the tag fields are also partially ordered [27] if we define the partial ordering relation such that given tags ’a’, ’a.b’, ’a.b.c’, and ’a.d’ the relation is defined as follows. Let the ’less or equal’ relation $\leq$ be such that tag a.b is less than tag a, i.e. a.b $\leq$ a. Now we have a.b.c $\leq$ a.b $\leq$ a but a.b $\not\leq$ a.d. Similarly, a tag with a time limit is ’lesser’ than a tag without any time limitation. Now tag ’hut’ gives larger access privileges than tag ’hut.cs-dept’, but tags ’hut.cs-dept’ and ’hut.as-dept’ are incomparable.

6.6 An Example of an Access Control Decision

Consider an example where the user comes to the door and presents the certificate chain she has. The message passing between the objects is presented in Figure 6.8. First the user presents her certificate chain to the door which asks the AccessControlDecisionDaemon to check whether this certificate chain provides access or not. The daemon instantiates an SPKIServices object with `checkPermission()` method (message 3 in the Figure). This method call takes for arguments[7] the `Tag` of the user certificate that the user presented, and an array of `SPKICertificate` objects, i.e. the chain, which should form the certificate chain described early on Figures 4.2 and 5.3.

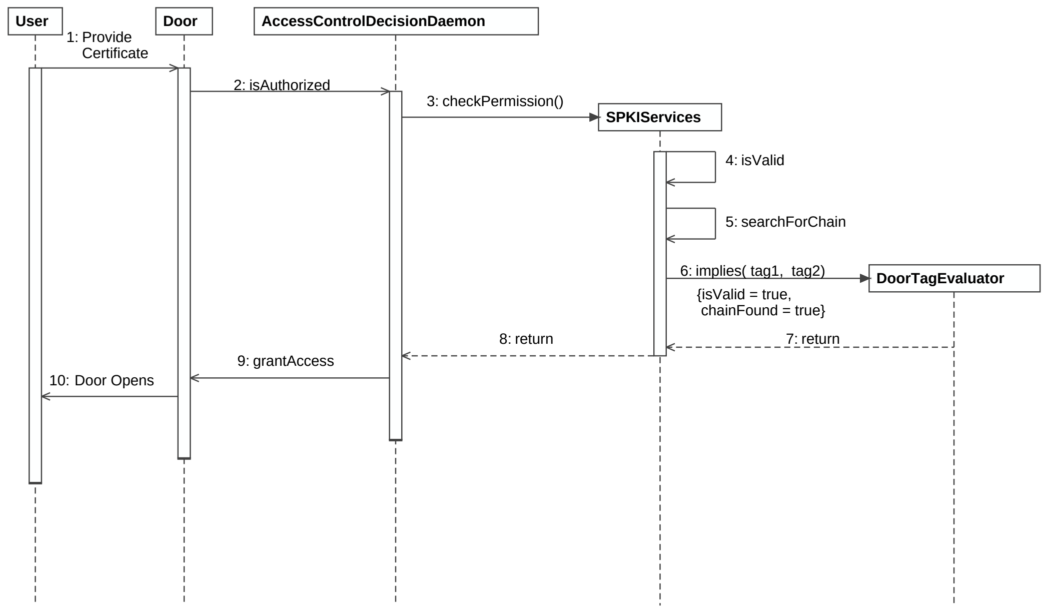


Figure 6.8: Access control decision making when an authorized user comes to the door

[7]We decided to omit the arguments from the figure in order to make the general situation clearer.

First, the `SPKIServices` checks whether the user certificate is valid. If it is, the next thing to check is whether the chain is complete. If it is, the final thing is to check whether the user has enough high clearance. `PermissionEvaluator` interface specifies a boolean method `implies(tag1,tag2)` which test whether permission specified in the first tag implies the permissions of the second. In our case, the first tag is the tag in the user's certificate and the second is the tag in the certificate of the door. If the first tag implies or more like dominates the second tag, the access is granted. In this scenario, `implies()` returns true which flows in steps 7–9 (Figure 6.8) to the door which grants access and opens.

Chapter 7

Evaluation and Conclusions

7.1 TeSSA Architecture

The origins of TeSSA lay in Nikander's vision about how to delegate access permissions and how to model trust in a distributed system [54]. Traditional, physical access control management systems are based on more or less centralized administration. An example of such a solution is the physical access control system currently being used in our university. We have a central Facility Management and Security (FMS) which ultimately administers the key distribution and policy definition. Rumors tell that some times there have been arguments between the university researchers and the FMS concerning the access to the university. Researchers wanted to have an unlimited 24-hour 7-day access and FMS wanted to restrict the access to office hours only, i.e. starting at 8 am and ending at 4 pm.

This is a concrete example of how the end-users' or departments needs, compared to the needs and policies of the central administrations, may sometimes differ drastically from each other. By applying TeSSA architecture we believe that the aforementioned problem could be solved. However, technology itself cannot solve all problems. A meaningful and realistic security policy is also needed. When we acquired both our physical and magnetic keys to the university, the laboratory engineer wondered - quite philosophically what does the professors do with 24-hour 7-days magnetic keys. They most usually have families to spend time with in the evenings. It is the students who work days and nights who are the actual users of the 24-hour 7-day a week access permission.

Defining the local security policy which would satisfy both the needs of the users *and* the restrictions which come from the general organization-wide policy is a difficult job. Our solution cannot solve that problem because it is not only technical in nature. Yet our solution is a step forward from a centralized solution because it distributes also the policy definition and management as much as it is technically possible to do.

7.2 Java and Design Tools

The design goals of Java programming language had been set quite ambitiously. The support for security features, networking, multi-threading, various extension APIs, and graphical user interface widgets have made it a natural choice for a programming language not just for us but almost anybody. Platform-independence has worked out quite well. The most obvious problem with different platforms in Java has been the file access. Java has not succeeded in hiding the different ways to address file names and paths. The structure of file systems in Windows environment and various UNIXs is so different from each other that the difficulty of hiding the differences is nearly impossible. Therefore the platform dependent features in Java file handling is rather well understandable.

Finding a good UML design tool was also quite problematic. We have already listed ArgoUML, Dia, and MagicDraw as the different tools which we tested. The principal factor on which we based on our evaluation was the completeness of the support for the various UML diagrams. We found out that the suppport was quite incomplete in various tools. The second factor was the requirements for the hardware which the application stated. ArgoUML and MagicDraw have been implemented in Java which placed quite high requirements on both the RAM and processor. We abandoned ArgoUML just because of the sluggishness of the implementation. Dia was quite good when it comes to the speed of the implementation. This is understandable since Dia was implemented in C or C++. However, the class diagrams looked extremely rough and coarse. Combined with the total lack of code generation possibility from the class diagram we abandoned Dia aswell. The only alternative left was MagicDraw which was good enough for us. Although it was quite slow to start, it was rather quick to use after the startup. The quality of UML diagrams and number of features it offered for the designer made it fairly good tool in general.

The GUIs were designed and implemented with a VisaJ[1]. VisaJ is a rapid application development tool which is also implemented in Java. It had a full support of Java Foundation Classes and Java 2 platform which made it an interesting tool to take a closer look at. Although it was nearly impossible to learn to use without taking a tutorial lesson with the manual, it turned out to be a good tool for GUI design. The code it generated could be further modified and at the same time the GUI could be modified also. It detected the sections we had modified and regenerated the GUI parts correctly without destroying our modifications to the code.

7.3 Reaching the Design Goals

Since our solution is based on using the public keys as the piece of recognizing information of the user, no name or other identifying information of the user is ever

[1]`http://www.ist.co.uk/`

revealed. Further, since we do not log authorized access decisions, traces or log files of comings and goings of the users are left nowhere. Usually there is no justified reason to log authorized entering. Since the public key is directly bound to the authority in the SPKI certificate, we do meet the requirement of authorization of actions without any identification. Also the privacy of the user is preserved since no logs are generated. The Finnish law concerning personal information registers requires that only legitimated personnel can access the personal data [43]. The organization-wide security policy should comply with the law and therefore all information which bounds the public key and the certificate to the person's name should be kept in safe hands and not visible to unauthorized persons.

Since the local security policy definition is distributed to the lowest level of the organization structure, there is no central authority who has to dictate the whole security policy. Further, in our solution, the user has the complete chain of certificates which she can present to the door. Therefore the door does not need to start collecting the certificates from any database while the access decision is under way. Thus we meet the requirement of distribution.

The local security policy manager defines the validity period of the user certificate. Therefore, the certificate becomes invalid sometime in the future and cannot be used after the validity is exceeded. Thus, the problem of non-returned physical keys does not exist. Our prototype does not specify the medium in which the user certificates are stored. If it was a cellular phone, we could not forbid the user from borrowing the phone to somebody else. However, since there is the risk that the borrower would make expensive calls with our phone, it is quite unlikely that the phone would be borrowed in the first place. There is no absolutely secure way to prevent the key borrowing - unless the key is strongly bound to us like something embodied, as we discussed earlier.

Since our implementation is still incomplete, we cannot make any measurements yet. The efficiency of our prototype cannot be measured yet. However, we tried to make some decisions on the design phase to meet the efficiency requirements as well as possible. The decision for better efficiency was to require that the user carries the whole certificate chain with her. If the door had to search for the certificates from DNS [31], it would be quite certain that the sub-second response time would be overrun.

7.4 General Considerations

Most of the companies today are heading for electronic commerce in the Internet. The majority of recent commercial solutions for doing so called 'secure electronic commerce' are based on identifying the parties. The X.509 has become a de-facto standard of security in commercial solutions. Some manufacturers sell their products by stating that “Electronic transactions and commerce will not work without secure identification” [21]. However, there is no proof that the person's name would be the inescapable property to ensure security.

We have studied and showed that physical access control can be designed and implemented by using TeSSA architecture without using names or similar identities at all. Although the implementation is still incomplete, the formal description of certificate chains in Chapter 5.5 and security labels in Chapter 2, Section 2.10 already proves that the concept is applicable.

The most important contributions of this work are the distribution of the management task and the detachment of identification and authorization from each other as distinct and separate problems. The distributed management provides better tools for administering the physical access control. The plain identity reveals very little if nothing either of the general permissions or the trustworthiness of a person. Therefore, the identity cannot be the relevant property when authorization decisions are in question.

However, there might be some environments or tasks requiring so extremely high security precautions that the identification is also necessary as part of the physical access control. Some military applications are an example.

Another aspect is that in our current solution there is no way to revoke the certificate during its validity time after it is issued to the user. Using our general solution, it is possible but the functionality is not implemented in the prototype. There are two different options. First, the certificate could be issued for only a short period, like a month at a time. When the time is running up, the end-user device could automatically fetch a new certificate. If there is a reason to not let the certificate be updated, then the automatic upgrade would not work. This could be a situation if the user was given a notice. The second alternative is to add an online test in addition to the validity information. In every leaf-level unit, there could be an additional online server which to ask whether the certificate still is valid in addition to the plain time constraint. Such an additional security measurement could be useful in an environment where there is high requirements for physical security an access control.

7.5 Further Work

Since Java bytecode is interpreted at run-time, it is clear that a native binary code implementation would be much faster in a real situation. A native code implementation is needed for testing the final efficiency and response time of the doors when they are doing the AC decision. It is highly probable that end-users will never accept a solution in which it takes seconds for doors to open.

Another aspect is the usability of the management GUIs. Our GUIs were designed for a prototype usage only without any user tests and further evaluations. More research is needed in the usability area. The visualization of security labels is another difficult task. It needs more studying.

A prototype of some commercial application based on TeSSA architecture would also prove that there is no actual need for identities in electronic commerce. The storekeeper's principal interest is to have the payment for the goods, not the identity of the customer. Nobody asks for our identity today when we buy our daily groceries. Why should anybody ask it on the Internet either?

Bibliography

[1] Edward Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall, 1994.

[2] Ross Anderson and Roger Needham. Robustness principles for public key protocols. In Don Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference*, volume 963 of *Lecture Notes in Computer Science*, Santa Barbara, California, USA, August 27–31 1995. Springer.

[3] Anon. Yleisimmät nimet. `http://www.compuline.fi/ComDocs/Suomi/uushlp/html/fin-0l2s.htm`, September 1999. [referred 11.3.2001].

[4] Network Associates. Introduction to Cryptography. `ftp://ftp.funet.fi/pub/crypt/cryptography/pgp/pgpi/6.5/6.5.1/linux/PGPf%reeware6.5.1-linux.tar.gz`, June 1999. (documentation included in `PGPfreeware6.5.1` distribution).

[5] Arto Astikainen / Helsingin Sanomat. Rakkauskirje-virus tuhosi kuvia ja sotki sähköpostijärjestelmät. `http://www.helsinginsanomat.fi/uutiset/juttu.asp?id=20000505TA30&pvm=20%000505`, May 2000. (in Finnish, referred 11.10.2000).

[6] Richard Barry and Duncan Campbell. Echelon: Proof of its existence. `http://www.zdnet.co.uk/news/2000/25/ns-16261.html`, June 29, 2000.

[7] M. Blaze, J. Feigenbaum, and J. Ioannidis. The KeyNote Trust-Management System Version 2, September 1999. Request For Comments 2704. `ftp://ftp.funet.fi/pub/doc/rfc/rfc2704.txt`.

[8] Matt Blaze, Joan Feigebaum, John Ioannidis, and Angelos D. Keromytis. The Role of Trust Management in Distributed Systems Security. In Jan Vitek and Christian D. Jensen, editors, *Secure Internet Programming*, volume 1603 of *Lecture Notes in Computer Science*, pages 185–212. Springer-Verlag, 1999.

[9] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures. In Bruce Christianson, Bruno

Crispo, William S. Harbison, and Michael Roe, editors, *Proceedings of the Security Protocols, 6th International Workshop*, volume 1550 of *Lecture Notes in Computer Science*, Cambridge, UK, April 15-17 1998. Springer.

[10] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *IEEE Conference on Security and Privacy*, Oakland, CA, May 1996.

[11] Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance Checking in the PolicyMaker Trust-Management system. In *Proceedings of the 2nd Financial Crypto Conference*, volume 1465 of *Lecture Notes in Computer Science*, Berlin, 1998. Springer.

[12] Mike Bremford. The Ping o' Death Page. `http://www.pp.asu.edu/support/ping-o-death.html`, November 1996. [referred 12.10.2000].

[13] Encyclopaedia Britannica. Lock, early history. `http://www.britannica.com`, 2000.

[14] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Spesification, December 1995. Request for Comments 1883. `ftp://ftp.funet.fi/pub/doc/rfc/rfc1883.txt`.

[15] D. Eastlake and O. Gudmundsson. Storing Certificates in the Domain Name System (DNS), March 1999. Request For Comments 2538. `ftp://ftp.funet.fi/pub/doc/rfc/rfc2538.txt`.

[16] Carl M. Ellison. What do you need to know about the person with whom you are doing business. `http://world.std.com/~cme/html/congress1.html`, October 1997.

[17] Carl M. Ellison, Bill Franz, Butler Lampson, Ronald L. Rivest, Brian M. Thomas, and Tatu Ylönen. Simple Public Key Certificate. Internet draft (expired), IETF SPKI Working Group, 26 July 1999.

[18] Carl M. Ellison, Bill Franz, Butler Lampson, Ronald L. Rivest, Brian M. Thomas, and Tatu Ylönen. Spki Examples. Internet draft (expired), IETF SPKI Working Group, March 1998.

[19] Carl M. Ellison, Bill Franz, Butler Lampson, Ronald L. Rivest, Brian M. Thomas, and Tatu Ylönen. SPKI Certificate Theory, September 1999. Request For Comments 2693. `ftp://ftp.funet.fi/pub/doc/rfc/rfc2693.txt`.

[20] Tommi Elo. Implementing ECDSA on a Java Smart Card. Master's thesis, Helsinki University of Technology, 2000.

[21] Mattias Erkkilä. Teknologia ei takaa luottamusta. Tietoviikko n:o 34, 05.10.2000. (in Finnish).

[22] S. Farrell and R. Housley. An Internet Attribute Certificate Profile for Authorization. `http://www.ietf.org/internet-drafts/draft-ietf-pkix-ac509prof-05.txt`, August 2000. Internet Draft (expires in six months).

[23] Jalal Feghhi, Jalil Feghhi, and Peter Williams. *Digital Certificates - Applied Internet Security*. Addison-Wesley, 1999.

[24] Joan Feigenbaum. Towards an Infrastructure for Authorization (Position Paper). In *3rd USENIX Workshop on Ecommerce Conference*, Boston, Massachusetts, Aug 31 – Sep 3 1998. USENIX. (not included in printed version of the proceedings).

[25] Joan Feigenbaum. Overview of the AT&T Labs Trust-Management Project. In *1998 Cambridge Univ. Workshop on Trust and Delegation*, volume 1550 of *Lecture Notes in Computer Science*, pages 45–58, Berlin, 1999. Springer.

[26] Martin Fowler and Kendall Scott. *UML distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2nd edition, 2000.

[27] Dieter Gollmann. *Computer Security*. John Wiley & Sons Ltd., 1999.

[28] Eric Greenberg. *Network Application Frameworks - Design and Architecture*. Addison-Wesley, 1999.

[29] Ralph P. Grimaldi. *Discrete and Combinatorial Mathematics, an Applied Introduction*. Addison-Wesley, 3rd. edition, 1994.

[30] N. Haller and R. Atkinson. On Internet Authentication, October 1994. Request For Comments 1704. `ftp://ftp.funet.fi/pub/doc/rfc/rfc1704.txt`.

[31] Tero Hasu. Storage and Retrieval of SPKI Certificates Using the DNS. Master's thesis, Helsinki University of Technology, 1999.

[32] Juho Heikkilä and Markku Laukka. SPKI Based Solution to Anonymous Payment and Transaction Authorization. In *Proceedings of the fourth Nordic Workshop on Secure IT systems (Nordsec'99)*, Kista, Sweden, November 1-2 1999.

[33] D. Ian Hopper. 'ILOVEYOU' computer bug bites hard, spreads fast. `http://www.cnn.com/2000/TECH/computing/05/04/iloveyou.01/index.html`, May 2000. [referred 11.10.2000].

[34] Cay S. Horstmann and Gary Cornell. *Core JAVA 1.2 Volume I - Fundamentals*, volume 1. Sun Microsystems Press, A Prentice Hall Title, 1999.

[35] Cay S. Horstmann and Gary Cornell. *Core JAVA 2 Volume II - Advanced Features*, volume 2. Sun Microsystems Press, A Prentice Hall Title, 2000.

[36] Sun Microsystems Inc. JDBC Data Access API. `http://java.sun.com/products/jdbc/index.html`, 1995-2001. [referred 12.3.2001].

[37] Internet Software Consortium. Internet Domain Survey Host Count. `http://www.isc.org/ds/hosts.html`, January 2000.

[38] Internet Software Consortium. Internet Domain Survey, January 2000: Number of Hosts advertised in the DNS. `http://www.isc.org/ds/WWW-200001/report.html`, January 2000.

[39] ITU-T recommendation X.509 (1997 E) information Technology - Open Systems Interconnection - The Directory: Authentication Framework, June 1997.

[40] Arto T. Karila. *Open Systems Security - an Architectural Framework*. PhD thesis, Helsinki University of Technology, March 1991.

[41] Kristiina Karvonen. Creating Trust. In *Proceedings of the Fourth Nordic Workshop on Secure IT Systems (NordSec'99)*, Kista, Sweden, November 1999.

[42] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol, November 1998. Request For Comments 2401. `ftp://ftp.funet.fi/pub/doc/rfc/rfc2401.txt`.

[43] Hekilötietolaki, n:o. 523/1999. (in Finnish, available on-line in `http://finlex.edita.fi`).

[44] Ilari Lehti. SPKI-based Access Control Server. Master's thesis, Helsinki University of Technology, 1998.

[45] D. Lewis and A. J. Weigert. Social Atomism, Holism and Trust. *The Sociological Quarterly*, 26(4):455–471, 1985.

[46] Ninghui Li, Benjamin Grosof, and Joan Feigenbaum. A Practically Implementable and Tracktable Delegation Logic. In *IEEE 2000 Symposium on Security and Privacy*, Oakland, California, May 2000.

[47] D. Maughan, M. Schertler, M. Schneider, and Turner J. Internet Security Association and Key Management Protocol (ISAKMP), November 1998. Request For Comments 2408. `ftp://ftp.funet.fi/pub/doc/rfc/rfc2408.txt`.

[48] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. Discrete Mathematics and its Applications. CRC Press, 1997.

[49] Jouni Meriläinen. Palvelunesto. Thesis written in course Tik-110.401 Fundamentals in Computer Security fall 1997. `http://www.hut.fi/u/merlin/dos/` (in Finnish). [referred 21.6.1999].

[50] Nicholas Morehead / Wired News. U.S. Endorses New Crypto Regs. `http://www.wired.com/news/politics/0,1283,37617,00.html`, July 2000. [referred 19.10.2000].

[51] Pekka Nikander. Modelling of cryptographic protocols – a concurrency perspective. Licensiate's thesis, Helsinki University of Technology, December 1997.

[52] Pekka Nikander and Kristiina Karvonen. Users and Trust in Cyberspace. In *Cambridge Security Protocols Workshop 2000*, LNCS, Cambridge University, April 2000.

[53] Pekka Nikander, Yki Kortesniemi, and Jonna Partanen. Preserving privacy in distributed delegation with fast certificates. In H. Imai and Y Zheng, editors, *Proceedings of Public Key Cryptography - Second International Worksho on Practice and Theory in Public Key Cryptography, (PKC'99)*, volume 1560 of *Lecture Notes in Computer Science*, Kamakura, Japan, March 1999. Springer.

[54] Pekka I. Nikander. *An Archtecture for Authorzation and Delegation in Distributed Object-Oriented Agent Systems*. PhD thesis, Helsinki University of Technology, March 1999.

[55] Office of the United Nations High Commissioner for Human Rights. Universal Declaration of Human Rights. `http://www.unhchr.ch/udhr/index.htm`, December 10, 1948.

[56] Kathleen Ohlson. Melissa: The Day After. `http://www.pcworld.com/news/article.asp?aid=10336`, March 1999. [referred 11.10.2000].

[57] Taloustutkimus Oy. Internet Tracking - Awereness and usage in Finland. `http://www.toy.fi/tuotteet/internet/inet5e.htm`, November 2000.

[58] Joon S. Park and Ravi Sandhu. RBAC on the Web by Smart Certificates. In *4th ACM Workshop on Role Based Access Ccontrol*, pages 1–9, Fairfax, VA USA, October 1999. ACM.

[59] Joon S. Park and Ravi Sandhu. Smart Certificates: Extending X.509 for Secure Attribute Service on the Web. In *22nd National Information Systems Security Conference (NISSC)*, Crystal City, Virginia, USA, October 1999.

[60] Jonna Partanen. Using SPKI certificates for access control in java 1.2. Master's thesis, Helsinki University of Technology, 1998.

[61] Nicholas Petreley. Linux users unscathed by ILOVEYOU. `http://www.cnn.com/2000/TECH/computing/05/09/linux.immune.idg/index.htm%l`, May 2000. [referred 11.10.2000].

[62] Ken Phillips. Biometric identification looms on landscape of network log-ins. `http://www.zdnet.com/eweek/reviews/0324/24bio.html`, March 1997. [referred 20.10.2000].

[63] Jon Postel (ed.). DoD standard internet protocol, January 1980. Request for Comments 760. `ftp://ftp.funet.fi/pub/doc/rfc/rfc760.txt`.

[64] Allen Walker B.Litt (Oxon.) (Chairman of Editorial Advisory Board) Read, editor. *The New International Webster's Comprehensve Dictionary of the English Language*. J.G. Ferguson Publishing Company, Chicago, Illinois, deluxe Encyclopedic Edition edition, 1996.

[65] Ronald Rivest and Butler Lampson. Cryptography and Information Security Group Research Project: A Simple Distributed Security Infrastructure (SDSI). `http://theory.lcs.mit.edu/~cis/sdsi.html`, 1996. [referred 1.11.2000].

[66] Ravi Sandhu and Pierangela Samarati. Authentication, Access Control, and Audit. *ACM Computing Surveys*, 28(1), March 1996.

[67] Ravi S. Sandhu. Lattice-Based Access Control Models. *IEEE Computer*, 26(11):9–19, November 1993. (cover article).

[68] Ravi S. Sandhu. Role-Based Access Control. In *10th Annual Computer Security Applications Conference*, 1994. `http://www.list.gmu.edu/articles/advcom/a98rbac.ps`.

[69] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.

[70] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 2nd edition, 1996.

[71] Simon Singh. *The Code Book - The Evolution of Secrets from Mary Queen of Scots to Quantum Cryptography*. Doubleday (a division of Random House, Inc.), 1st edition, 1999.

[72] Douglas R. Stinson. *Cryptography: Theory and Practise*. Discrete Mathematics and its Applications. CRC Press, 1995.

[73] Suomen perustuslaki, n:o. 731/1999. (in Finnish, available on-line in `http://finlex.edita.fi`).

[74] Sanna Suoranta. An object-oriented implementation of an authentication protocol. Master's thesis, Helsinki University of Technology, November 1998.

[75] Teija Sutinen. Joukkoliike yksityisyyden puolesta. Helsingin Sanomat, `http://www.helsinginsanomat.fi/uutiset/juttu.asp?id=20000912TA1&pvm=200%00912`, 12.9. 2000. (in Finnish). [referred 11.10.2000].

[76] Teppo Tiilikainen / Helsingin Sanomat. Vaarallinen virus saastuttaa tietokoneita ympäri maailman. `http://www.helsinginsanomat.fi/uutiset/juttu.asp?id=990329ta07&pvm=1999%0329`, March 1999. (in Finnish). [referred 11.10.2000].

[77] VeriSign. VeriSign Security Alert Fraud Detected in Authenticode Code Signing Certificates March 22, 2001. `http://www.verisign.com/developer/notice/authenticode/index.html`, March 2001. [referred 26.3.2001].

[78] Väestörekisterikeskus. Yleisimmät sukunimet väestötietojärjestelmässä. `http://www.vaestorekisterikeskus.fi/sukunim.htm`, Maaliskuu 1998 [referred 2.3.2001].

[79] Teemupekka Virtanen. Tutkimus tietoturvallisuuden inhimillisistä tekijöistä. Licensiate's thesis, Helsinki University of Technology, December 1992.

[80] M. Wahl, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3), December 1997. Request For Comments 2251. `ftp://ftp.funet.fi/pub/doc/rfc/rfc2251.txt`.

[81] Object Management Group (OMG) Website. `http://www.omg.org`. [referred 9.3.2001].

[82] C. Weider and J Reynolds. Executive Introduction to Directory Services Using the X.500 Protocol, March 1992. Request for Comments 1308. `ftp://ftp.funet.fi/pub/doc/rfc/rfc1308.txt`.

[83] Gregory B. White, Eric A. Fisch, and Udo W. Pooch. *Computer System and Network Security*. CRC Press, 1996.

[84] Phil Zimmermann. Homepage, Background of Phil Zimmermann. `http://www.pgp.com/phil`. [referred 11.10.2000].

[85] Phil Zimmermann. Homepage, Letters to Phil. `http://www.pgp.com/phil`. [referred 11.10.2000].

[86] Philip Zimmermann. PGP User's Guide. `http://www.tml.hut.fi/Opinnot/Tik-110.350/Tehtavat/pgp/`, October 1994. [referred 11.10.2000].