Decentralized Authorization - a survey

Licentiate's Thesis

Sanna Liimatainen

Helsinki University of Technology
Department of Computer Science and Engineering
Telecommunications Software and Multimedia Laboratory
Espoo 2002

Teknillinen korkeakoulu
Tietotekniikan osasto

Tietoliikenneohjelmistojen ja multimedian laboratorio

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF LICENTIATE'S THESIS

Author: Sanna Liimatainen

Name of thesis:
Decentralized Authorization - a survey

Date: Autumn 21, 2002 **Pages:** 10 + 135

Department: Department of Computer Science and Engineering **Chair:** T-110

Supervisor: Professor Teemupekka Virtanen
Instructor: Docent, D.Sc. (Tech) Pekka Nikander

Currently used centralized, identification and authentication based security systems do not scale well in heterogeneous computing environment. Other common problem for all security systems is usability. The systems are based on complex technology and there are no working metaphors for functioning in the systems.

Better solutions for trust management and access control in heterogeneous computing environment are decentralized authorization systems. These systems suites well even for electronic commerce.

This work introduces a comparison methodology based on Common Criteria and heuristic evaluation usability method. The basic functions of decentralized authorization systems are identified with Common Criteria. Based on these basic functions, central tasks of users in different decentralized authorization systems are described. The usability of the systems is evaluated with heuristic evaluation.

The decentralized authorization systems evaluated in this work are PolicyMaker and KeyNote trust management systems, Simple Distributed Security Infrastructure / Simple Public Key Infrastructure (SDSI/SPKI), and Telecommunications Software Security Architecture (TeSSA).

Keywords: decentralized authorization, trust management, usability
Language: English

TEKNILLINEN KORKEAKOULU

LISENSIAATINTYÖN TIIVISTELMÄ

Tekijä: Sanna Liimatainen
Työn nimi: Hajautettu valtuutus

Päivämäärä: 21. elokuuta 2002 **Sivuja:** 10 + 135

Osasto: Tietotekniikan osasto **Professuuri:** T-110

Työn valvoja: Professori Teemupekka Virtanen
Työn ohjaaja: Dosentti, TkT Pekka Nikander

Nykyisin käytetyt keskitetyt, tunnistamiseen ja todentamiseen perustuvat tietoturvajärjestelmät eivät skaalaudu hyvin heterogeenisessä tietokoneympäristössä. Toinen näille järjestelmille yhteinen ongelma on käytettävyys. Järjestelmät perustuvat monimutkaiseen tekniikkaan ja sopivia metaforia järjestelmissä toimimiselle ei ole.

Parempia ratkaisuja luottamuksen hallintaan ja pääsynvalvontaan heterogeenisessä ympäristössä ovat hajautetut valtuutusjärjestelmät. Ne sopivat hyvin jopa sähköiseen kaupankäyntiin.

Tämä työ esittelee vertailumetodologian, joka perustuu Common Criteriaan ja heuristisen evaluoinnin käytettävyysmenetelmään. Hajautettujen valtuutusjärjestelmien perustoiminnot on tunnistettu Common Criterian avulla. Perustuen näihin toimintoihin, keskeiset käyttäjän tehtävät on kuvattu erilaisissa hajautetuissa valtuutusjärjestelmissä. Järjestelmien käytettävyyttä on arvioitu heuristisen evaluoinnin avulla.

Tässä työssä arvoidut hajautetut valtuutusjärjestelmät ovat PolicyMaker ja KeyNote luottamuksenhallintajärjëstelmät, Simple Distributed Security Infrastructure / Simple Public Key Infrastructure (SDSI/SPKI) ja Telecommunications Software Security Architecture (TeSSA).

Avainsanat: hajautettu valtuutus,luottamuksen hallinta, käytettävyys
Kieli: englanti

Acknowledgements

My instructor, Pekka Nikander has helped me a lot in writing this thesis. I thank him for giving ideas and framing. I thank professor Hannu Kari for his help for making a working comparison methodology. Thanks to professor Arto Karila and professor Teemupekka Virtanen for allowing me to continue studies in an interesting field of computer science.

I want to thank my friends, Ursula Holmström and Kristiina Karvonen, for their valuable comments. I also thank Tero Hasu, Jan Hlinovsky, Janne Lindqvist and Mika Ranta for their help.

Finally, I thank my family. Especially my husband Janne has supported me and taken care of me and everything during this work.

Thanks to all!

Otaniemi, August 21st, 2002

Sanna Liimatainen

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Security is a necessary component of almost every computing system. Security issues becomes complex when the system grows. For example, the physical protection of the system is not enough when the parts of the systems are connected through a network. Valuable information, expensive resources, and precious services must be protected from disclosure, illegal changes, and misuse.

Currently used security management systems are based on identification and authentication. The system keeps up a list of names of the authorized users, and the user identifies herself to the system in order to get access to information, resource, or service. Basis of these systems lies on governmental way of doing business: a citizen is identified based on her identification number before she can even explain her problem. Eventually, this two phase method of identification and authentication is clumsy and often unnecessary.

Better security management solutions are based on authorization. Often the identity of the user is unessential information. The system needs only to know that the user has authorization to use the system and not who the user really is. Other great benefit of authorization systems is that the issuer of the authorization is often also the entity who finally decides weather the user is authorized to perform the requested action, i.e. external trusted third parties are unnecessary.

One of the biggest problems of security systems is that they are not easy to use. Security solutions have complex mathematical background and the systems are often complicated. The problem of unusable security slows down the electronic commerce and other businesses on top of an unsecured network such as Internet. Common users and companies do not trust to the security solutions because they do not understand how the solutions works.

A solution, that is sometimes suggested, is that the security is hidden from the users. This is not a good choice because when the user is completely uninformed about the underlying security, she cannot base her trust towards the system on knowledge.

This thesis compares different solutions that provide decentralized authorization from usability point of view. The purpose is to identify usability problems of decentralized authorization mechanisms.

Organization of this Thesis

The chapters are organized as follows. In the second chapter, the background for security management information is introduced. Firstly, the basis of public key cryptography, and secondly, the basis for currently used public key infrastructures are described. The second chapter discusses also why authentication should be replaced with authorization, and how authentication and authorization works in a decentralized system.

The third chapter gives different kinds of aspects for the comparison: usability, applicability, and scalability.

The fourth chapter describes five different kinds of approaches to solve decentralized authorization. The first decentralized trust management system is PolicyMaker. Other solution, Keynote, is based on same kind of architecture than PolicyMaker. Other kind of solution is a PKI called Simple Public Key Infrastructure. Last decentralized authorization system described in this thesis is the TeSSA architecture.

The fifth chapter describes the comparison methodology that is based on Common Criteria and two usability test methods, heuristic analysis and cognitive walkthroughs. This chapter also introduces the test cases which are used in the comparison of the different decentralized authorization approaches.

Results of the comparison are presented in chapter six, as well as the experience gathered from the comparison.

Finally, the chapter seven concludes this work.

Chapter 2

Background

Service providers offer services for their customers. Developing and maintaining services cost money and service providers collect that money from the customers. In order to do that, a service usually identifies and authenticates its users or checks that a user has authorization to use the service. Several methods have been developed for providing authentication or authorization.

In a distributed system, computing and data are spread to several computers usually connected by a network. Problems are divided to smaller problems and divided among computers, i.e. a computer solves the part of the problem where it is the most efficient. Usually, the distributed system still has common administration. When a distributed system is no more under single administration, the system is decentralized. Its parts are independent but still working together. If one part of the system cannot answer to a question, it forwards it to others. For example, authorization information needed by a service provider can be issued by several different authorities, and the information must be gathered from different sources.

In this chapter, the background of decentralized systems and their security are introduced. Firstly, I present the techniques that are needed in public key infrastructures. The second section concerns authorization, and the last section introduces decentralized systems.

2.1 Public Key Cryptography

As we will see, public key cryptography and public key infrastructures (PKI) are the base technology decentralized authorization is built upon. In this section, Firstly, I introduce methods that are needed for establishing a public

key infrastructure. In the next section I describe what a public key infrastructure is and give examples of such.

The following names are commonly used for clarifying the roles of actors in a system based on public key cryptography: Alice, Bob and Carol are users that want to perform some actions. Eve is an evil user that usually wants to do something illegal or harmful. Some actions need external actors. One of such actors is Trent who is a trusted third party.

As an example, let us consider a situation where Alice wants to send a message over the Internet to Bob and wants to be sure that Eve cannot read or modify the message. Alice encrypts the message with an either symmetric or public key cryptographic algorithm to ensure confidentiality of the message. She also digitally signs the message to make sure that Eve cannot modify the message without being detected. Thereby Bob knows that the message is really coming from Alice, it has not been changed during transportation, and Eve has not read the message.

Alice needs a secure way to exchange encryption/decryption key (depending on the cryptographic algorithm) with Bob to ensure that the key really belongs to Bob. One possibility is that someone trusted, Trent, ensures Alice that Bob's public key is really his key. The function of a public key infrastructure is to ensure that users get the right keys.

Even though in the previous example a key is bound to an identity of a party, that is not always the situation. A public key infrastructure can bind also authorization to a key, that can be used to represent authority to do something. This chapter presents the traditional public key infrastructures which are mainly used for binding identity to a key.

In the following subsections, I first discuss basics of cryptographic techniques that are needed in PKIs and then more about PKIs and how they work.

2.1.1 Encryption and Decryption Algorithms

An encryption algorithm changes a plaintext message to an unreadable form, a ciphertext, and a decryption algorithm operates to the opposite direction. Both algorithms use keys. Figure 2.1 clarifies the situation where a message is transferred confidentially through an untrusted network. First, the encryption algorithm uses a key to change the plaintext to the ciphertext. The ciphertext can be transmitted over untrusted channel. The decryption algorithm changes the ciphertext back to the plaintext using either the same key or another key depending on the used algorithm.

Figure 2.1: Encryption and decryption

A keyspace is a large set of possible keys and the key is usually a long number picked out from the keyspace. In general, cipher algorithms are divided in two groups: symmetric and public-key algorithms.

Symmetric algorithms use same key in both encryption and decryption. Communicating peers need to securely agree on the key. If members of a group trust each others, they can use one common key. The possibility to cheat grows when the group becomes larger. For example, one member can give the key to an outsider who then can follow the communication. If a group of people want to communicate together in a secure way, they all need to agree keys with each other member of the group. To allow confidential communication between each pair of users in a group, n*(n-1)/2 keys are needed. The key agreement may also be problematic. Usually a key distribution center (KDC) is used to make key agreement more efficient.

In public-key cryptosystems, a pair of mathematically related keys is used. One key is used to encrypt data, and only its pair key can decrypt data, not even the encryption key reveals the original data. Usually, the encryption key is called a public key and the decryption key is called a private key. The public key can be published, for example, in a WWW page and everyone can use it to encrypt messages. Only the owner of the corresponding private key can decrypt the messages. Public key cryptosystems have their own problems. Algorithms are considerably slower than symmetric ones and keys are longer than in symmetric cryptosystems [78]. Thus, public-key cryptosystems are often used to protect a key exchange that creates a session key for symmetric cryptosystems. Then the actual messages are encrypted with the symmetric algorithm.

Public-key cryptosystems have one more problem: how can Alice know that the public key at Bob's WWW page is really Bob's key? The web page could be spoofed, anyway. One way is that the public key is digitally signed by Carol whose key Alice has got directly from Carol and to who Alice trust. This is one kind of simple PKI, but before we go deeper we look what other

cryptographic methods PKIs need.

2.1.2 Digital Signatures

A digital signature ensures that a message or a document is integral and that the signer is the originator of the document. Signing documents with public-key cryptography works in the opposite direction than encrypting with it: the originator signs the document by encrypting it with her private key and then everyone can verify the signature by decrypting the document with the originator's public key. In some public key systems, there is a separate key pair for signing and for encrypting. It is also possible to use a symmetric cryptosystem to sign documents but then an arbitrator, a trusted third party, is needed.

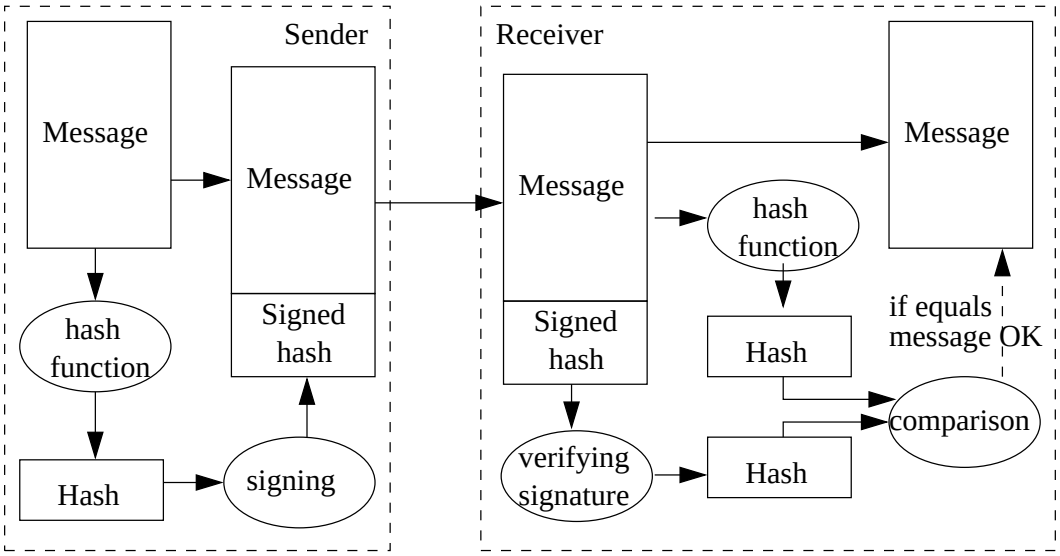


Figure 2.2: Signing and verifying a message

Like encrypting and decrypting with a public-key algorithm, also signing a large document with public-key cryptography is inefficient. Figure 2.2 illustrates more efficient method of signing documents or messages. Instead of signing a whole document only a hash calculated from the document is signed. A hash function compresses an arbitrary length message to fixed number of bits. Good hash functions are collision-free, i.e., it is not possible to find two messages that have same hash value. The signed hash is sent

together with the original message to the recipient. She verifies the signature by first calculating the hash from the original message and "decrypting" the signed hash and then comparing the hashes.

2.2 Public Key Infrastructures

Whitfield Diffie and Martin Hellman presented in their article the idea of public key cryptography in 1976 [23]. As I already mentioned, the main problem in public key cryptography is the users knowing that they have the key of the right recipient. A solution was suggested to form a list of name - public key pairs stored in a Public File, similarly to a telephone directory that contains name - telephone number pairs. In his bachelor thesis in 1978, Loren Kohnfelder stated that a Public File is not useful because fetching keys from it is not efficient and such a centralized system is hard to maintain [55]. He proposed the use of certificates in binding the public keys to names. The certificates could be issued by the administrator of the Public File and this way the first public key infrastructure was introduced.

The purpose of this most simple form of public key infrastructures is to publish the public keys used in public key cryptography. In other words, a PKI ensures that the public key belongs to the entity that is claimed to belong to and this entity has the corresponding private key. Basically, a PKI consists of two parts: certification and validation [19]. A certificate binds the public key to a name of a person or an entity, or even to some other information like a permission. Validation process verifies that the certificate is still valid in the moment of use. In the next subsections, certification and validation are discussed more widely with examples. Later in this thesis, other kinds of credentials that are used to denote authorization are discussed.

2.2.1 Certification

A certificate is a digitally signed record that states properties of some entity. A certificate attaches some information to a public key. Certificates can be divided into three groups based on that information: Identity certificates state the identity, a name, of the public key holder. Credential or authorization certificates express permissions or credentials. Attribute certificates are little different: they bind authorization to a name and they are used with identity certificates that bind the information to public keys.

In this section, I introduced these three kinds of certificates, what kinds of

structures there are for granting them, and what kind of certificate systems there are in use.

Certificate Types

When a word certificate is mentioned in the context of computer security, people usually think about identity certificates. They are the oldest form of public key certificates introduced by Kohnfelder in 1978 [55]. Identity certificates and certification authorities (CA) work in similar way than labels in French wines [48]. French Institut National des Appellations d'Origine (INAO) organization manages a classification system called Appellation d'Origine Contrôlée. This organization certifies that a wine bottle comes from the vineyard mentioned in the label of the bottle. Similarly to the labeling system, a certification authority assures that a public key belongs to an entity which name is mentioned in the certificate. There are two widely used identity certification systems: X.509 [47] and Pretty Good Privacy (PGP) [84]. Both of these are briefly introduced later in this section. Nevertheless, the Appelation d'Origine Contrôlée organization does not guarantee the quality of the wine and same way, the identity certificate do not guarantee that the person do have authorization to perform an action.

An other form of certificates are authorization certificates, which are often also called credentials. They bind an authorization to a public key. This authorization might be, for example, execution permission for a program downloaded from the Internet. Authorization certificates do not use names in the certificates but a permission is given directly to a public key. Thus, the owner of the corresponding secret key becomes a holder of the permission denoted in the certificate. Issuing authorization certificates is different from issuing identity certificates. There might not be an actual certification authority, but the owner of a resource can be and usually is the primary source of authorization.

Identity certificates are like passports. Sometimes, a passport is not enough for entering to a country, but an additional document, a visa, is needed. Attribute certificates are like visas [34]. An attribute certificate combines authorization to an identity. Attribute certificates are used together with identity certificates: An attribute certificate binds authorization to an identity, and an identity certificate binds identity to a public key.

Granting Certificates

There exits several different kinds of certificate granting schemes. X.509 identity certificates are usually used with hierarchical certification authorities and globally unique names. In PGP, an electronic mail address acts as an unique identifier of a user, and a friend of a user certifies that a public key really belongs to that user. Thus, a group of friends forms a "web of trust". Authorization certificates of Simple Public Key Infrastructure (SPKI) can form a certification loop that is originated from the owner of the resource and gives a right to use the resource to a user. Below are examples from each of these.

Examples

FINEID. Since December 1999, Finnish citizens have been able apply for an electronic identification (FINEID) card [20]. The Finnish Population Register Centre acts as a certification authority and issues the certificates. It also certifies services and servers. The FINEID card contains two X.509 certificates and RSA key pairs. One certificate is used for authenticating the user to services, and another certificate is used for encrypting and signing of documents. Services can be offered by authorities, communes, and private sector organizations such as banks. The Population Register Centre maintains a list of all the public keys of FINEID card holders and their public keys on its WWW pages. Certificates in the FINEID card binds a RSA public key to name of the card holder. Because several people can have exactly same name, an unique number called electronic identification number is also stored in the card. A problem arises, for example, when Alice want to send an encrypted message to Bob using public key verified by Population Register Centre. There might be several Bobs in the list and the only way to distinguish them is to know the electronic identification number.

PGP at HUT. In Telecommunications Software and Multimedia Laboratory (TML) at Helsinki University of Technology (HUT), Pretty Good Privacy (PGP) is used to authenticate students of basic courses when they return their homework assignment answers by electronic mail [59]. A student writes her answers to a file, signs it with PGP, and sends it to the course electronic mail address. Before sending answers of any basic course assignment, the student must physically authenticate herself to the course staff. The student brings a fingerprint (a hash) of her public PGP key printed in a piece of paper and shows her student card or other identification card to the course assistant. The assistant then signs the public PGP key of the student with

the PGP key of the course. When the same student enroll to another course in the TM-Laboratory, she does not need to authenticate herself again. Staff of this course can see that another course has verified that this key to belongs to this student. This way, the students and the courses form a structure "web of trust".

Authorizing Java code with SPKI. Java programming language has made mobile code reality: everyone can download a program from a WWW page and execute it in her own computer. Pekka Nikander and Jonna Partanen have introduced authorization certificate based system for authorizing mobile Java code [68, 72]. Their system makes execution of mobile code safe. They give an example to clarifies how the system works [68]: Alice as an administrator of a computer wants to give file access permissions to a program developed by Bob. Firstly, one certificate denotes that Alice has right to define any permissions for her computer. A second certificate issued by Alice gives to Bob a right to authorize programs to access files in Alices computer. In the third certificate, Bob gives the file access rights to his new program. These three certificates form a chain that gives Bob's program the right to access files in Alices computer. The program can proof with the certificates to the computer that it really has that right. Section 3.2.3 clarify more precisely how authorizing Java code with SPKI authorization certificates works and why such a system is necessary. The example above shows that everybody can act as certificate authority and issue authorization certificates. There is no actual trust hierarchy in the case of authorization certificates.

After the examples above, we can see that there must be some way to check that the certificates are valid in the moment of use. The next subsection shows different methods to check the validity of certificates and what kinds of problems there are in validation.

2.2.2 Validation

First of all, the receiver of a certificate must check that the certificate itself is not forged. This is done by checking the signature of the certificate. But it is not enough because the situation might have changed since the signing of the certificate. Some external validation of information is needed, and first in this subsection, I discuss offline and online validation. Then I introduce the concept of revocation and how certificate revocation lists (CRL) are used. Finally, last in this section I discuss the problems of public key infrastructures.

Online and Offline Validation

Offline validation means that validity information is stored in the certificate. Let us consider the examples about the different certification granting schemes from previous subsection. The FINEID identification card is valid for three years. After the three year time period, the keys stored in the card are still usable but they are not considered valid anymore. In the case of authorization certificates in the example above, Bob could have given the file access permissions to his program for just one day. After that day, the program does not have the right to access files in Alice's computer. As we can be see, the offline validation must be considered when the certificate is created. But what can be done if the public key in the certificate is compromised? This kinds of threats can be handled with online validation. The certificate also contains locations (URLs) of validation services and the public key that the service use to denote validation.

When buying an expensive thing with a credit card, the cash register will call to a validation number in order to check that the credit card is not on a list of stolen cards. Similarly, some certification schemes support online validation. A service can check from the certification authority that the used certificate is still valid. If a certificate is invalid, it has been revoked.

Certification Revocation List (CRL)

Pure online validation system requires much, for example, the service must be reachable all the time and it must be able to handle many simultaneous queries. Usually revoked certificates are collected to a certificate revocation list (CRL).

A certification authority can publish certificate revocation lists. A list is digitally signed by the CA to ensure that it is real. For example, the Finnish Population Register Centre publishes FINEID certificate revocation list every hour and the list is valid for two hours. This example illustrates one particular problem of CRLs: a certificate may need to be revoked just after the previous CRL has been published. Solutions such as delta-CRLs are developed for this problem, but they do not belong to the scope of this work.

2.2.3 Problems of PKIs

Public Key Infrastructures solves problem of knowing to whom a public key belongs or what kinds of actions a keyholder is authorized to perform. Still,

PKIs are far from being perfect solution. Carl Ellison and Bruce Schneier gives ten problems of PKIs [30]. The list consists of following items which some I introduce in more details:

- Necessity of PKI
- Trusting to a certification authority
- Authority of a certification authority
- Identification process before granting a certificate
- Human names are not global
- Secure practices to use certificates
- Protecting the private key
- Security of a certification authority, i.e. that it is implemented and secured with care
- Security of a verifying computer, i.e. it cannot be cheated or break into
- Security of an application, i.e. the application is working trustworthy and tell necessary information to the user

First of all, one should ask if a public key infrastructure is really necessary. Public key infrastructures has said to solve, for example, the problems in electronic commerce, but the currently used systems are not "beautiful". Most of the systems secure the underlying connection between a client machine and a vendor server , and uses the common practices from the non-electronic world. They does not really solve problems of electronic commerce.

A user must trust to a certification authority in two ways: that the certification authority is trustworthy and that it has authority to issue the authorization. For example, a widely used WWW security system is based on Secure Socket Layer (SSL) protocol defined by Netscape and X.509 identity certificates. Usually, a WWW client program gives a list of certification authorities that are implicitly trusted to certify servers. The user must decide if she trusts to Canadian post.

Problem of naming

Centralized systems does not scale well, and identity certificate systems such as X.509 is a centralized system. In this subsection I give one justification for the above claim. I describe more precisely later in section 3.3 what scalability means and what other problems makes a system non-scalable.

One of the main problem of X.509 identity certificates is naming. Names works well as identifiers in a family [30]. When names are used in global scale under one authority, a problem arises: human names are not globally unique. For example, the most common family name in Finland is Virtanen and the most common man's name is Juhani [21]. When Alice want to send secure electronic mail to Juhani Virtanen, a problem arises. How she can know which public key certified by a FINEID identity certificate described above belongs to that Juhani Virtanen she knows? Like describe earlier in section 2.2.1, some additional information such as an identification number, an address, or a working place can be used to distinguish the names.

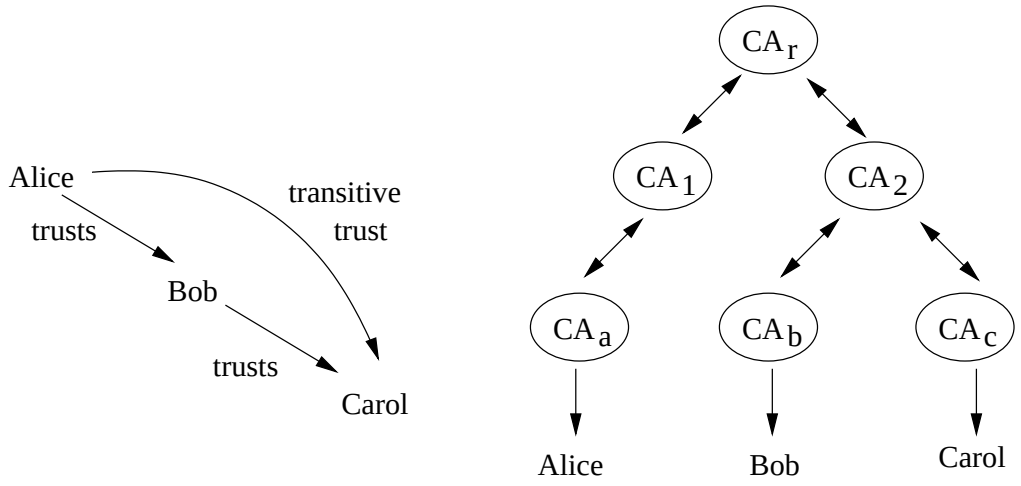
One solution for globally unique names is hierarchical naming as given in the example above. A name have two parts, the name of a person and some additional information like the organization where this person is working. Still, the additional information does not solve the problem. For example, when a person change her working place, the name of the organization will change.

Later in this thesis, in section 4.3.1, a solution for the naming problem is presented. The solution is base on linking of local names which is natural to humans.

Problems of Authorization PKIs

The problems listed above are mainly problems of such public key infrastructures that provide identity bindings. There are also problems in the authorization public key infrastructures. Decentralized systems have usually both human and non-human (such as computer programs etc) actors. Trust in computer world is considered to be transitive, but this is not true in human relationships [1]. For example in figure 2.3(a), if Alice trusts to Bob and Bob trusts to Carol, it does not necessary means that Alice trusts to Carol. On the other hand, this is just what delegation in the authorization certificate chain means. Of course, some restrictions to the trust can be defined.

As a matter of fact, trust is considered to be transitive also in the traditional identity certificates. The granting of certificates is based on a tree of



(a) Transitive trust (b) CA hierarchy

Figure 2.3: Transitivity of Trust

certification authorities like presented in figure 2.3(b). The CAs trust each other and the use of identification certificates assumes that all users trust to all CAs. In the example, Alice must trust to the certification authority CA_c that it has truly identified Carol with secure manners.

Public key infrastructure offers powerful tool for surveillance of individual acting on networks, which threaten privacy of people [45]. Traditional public key infrastructures uses identity more than is necessary. For example, in electronic commerce merchants do not need the true identity of a client, a pseudonym is enough for profiling purposes. Authorization certificates offer alternative where a user can have several keys for different purposes. SPKI authorization certificate documentation is even worried that a set authorization certificates can reveal identity of the certificate holder [27].

As denoted in the Introduction of this thesis, there are two kinds of public key infrastructures: one PKI binds a public key to a name and identifies the owner of the public key, and other PKI binds a public key to a right and authorizes the owner of the public key. In the next section, these two ways to express rights of a user are compared.

2.3 Authorization

Authentication and authorization are two basic parts of security systems. They can both be used when access control situations need to be solved. In this section, I first briefly check what is access control and authentication and then discuss why identity authentication should be replaced with authentication of direct authorization expressions.

The function of access control is to decide weather to let someone to use a resource or not. This information can be conceptually represented as an access control matrix [5]. Subjects (e.g. users) are listed in the rows of the matrix and objects (e.g. services) are columns of the matrix. How a subject can act upon an object is defined in the cell of the matrix. An access control list (ACL) is a simple representation of the access control matrix where each object is altered with a list of its legal users and their rights. That is, in addition to the users, the ACL can also contain more detailed information regarding what the subjects can do to the objects.

But access control as such is not enough. When a user want to use a service to which she has access rights, she first somehow has to show that she really is who she claims to be, i.e., authenticate her identity to the service. Traditionally, the user first identifies herself by giving a user name and then authenticates herself by giving a passphrase for the service. In this example, authentication is based on something known only by this user: the passphrase. Other forms of authentication are based on something embodied and something held [5]. Something embodied, for example, a fingerprint, is used long time for solving crimes. Something held is also an old concept in access control: keys have been used for opening locks for quite a long time.

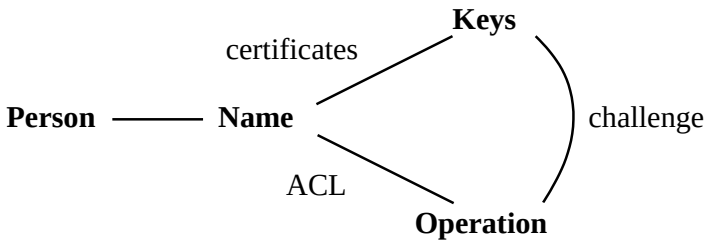


Figure 2.4: Identification certificate bindings [58]

Most of the services do not actually need the identity information about the user. Often a service simply needs to know if the user is authorized to do what

she wants to do. Joan Feigenbaum argues [35] that authentication should be replaced with authorization because then systems will become simpler. In traditional ACL and identity based systems, there are two questions to be answered: "does this user own the secret key corresponding to this public key" and "does the owner of the secret key have the authorization to perform an action". The figure 2.4 illustrates the bindings that an traditional public key infrastructure creates to be able to answer these question [58]. For the first question, a public key infrastructure can answer, for example, by giving a X.509 certificate that binds a name to a public key. The access control list describes what operations a person (the name) is authorized to perform. When the person want to perform the action, authorization is checked by challenging the public key.

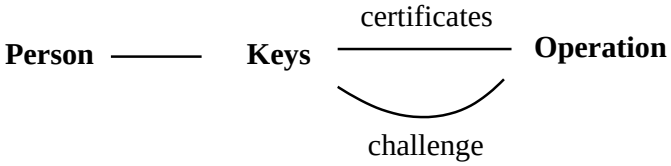


Figure 2.5: Authorization certificate bindings [58]

The first question above is irrelevant for the service. Actually, it needs an answer only to the second question. For example, when a company makes an electronic contract with someone, it is not so essential who the signer of the contract is but that she has the authorization to sign the contract [36]. Joan Feigenbaum states that "rather than supporting signed requests with name-key binding certificates, requesters should support them with more flexible and general credentials that prove that keys are authorized to take actions" [35]. The credential based certificate bindings is presented in the figure 2.5. The authorization is directly given to a public key in a certificate and the verification of authorization is also directly between the operation and the public key.

2.3.1 Certificate Chains and Loops

Authorization certificates may support delegation. Delegation means that a right given in a certificate can be further given to someone else in a new certificate. The certificates form a chain from the issuer of the first certificate to the subject of the last certificate. The chain is valid if all the certificates,

except the last one, give right to their subjects to further delegate the right given in the certificates. Of course, the issuers cannot give extra rights that they do not have to anyone. Certificates are closely related to trust relationships [58]. In a certificate chain, the trust is transitively propagated from one entity to the next entity and further on.

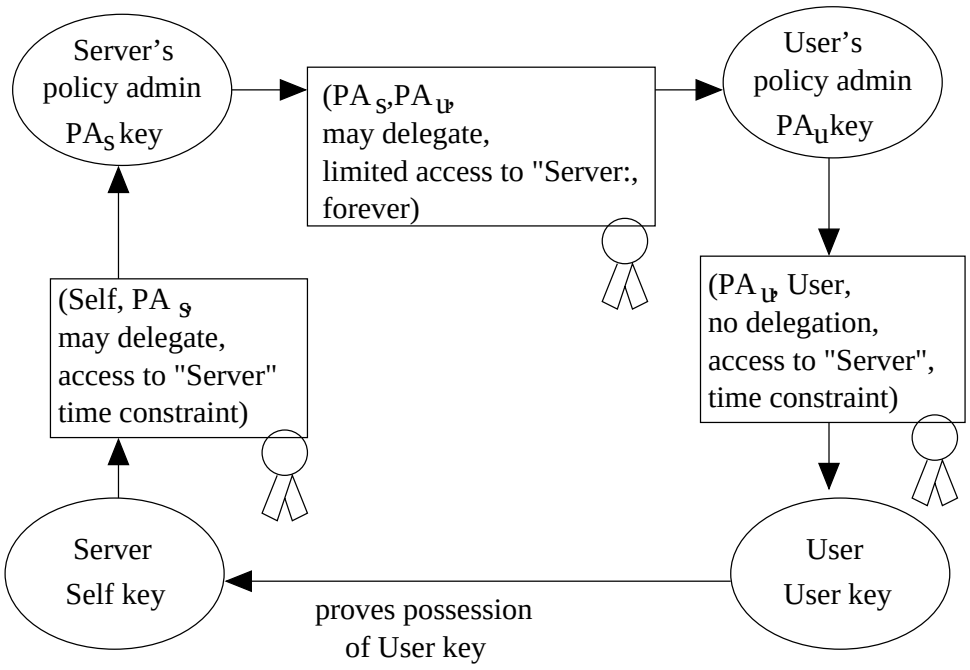


Figure 2.6: An authorization certificate loop [58]

Usually, an entity who has issued the first certificate of a certificate chain is also the entity who finally checks that the certificate chain is valid. Figure 2.6 presents a certificate loop that is formed from a certificate chain and a query. In the example of the figure, a server offering a resource issues a certificate to its administrator and gives the right to further delegate the authorization to use the resource. The administrator grants a certificate with limited access right to another administrator. This administrator grants a certificate with no delegation rights to a user. When the user wants to use the resource in the server, she gives to the server the certificates of the chain as a proof that she has the rights to use the system. The server checks that the chain is valid by firstly verifying the signatures of the certificates, and then checking that the chain is unbroken.

Certificate loops work also to the opposite direction. When a user want to

know that a server is the right server, another certificate loop is formed. The user gives certificate to a certificate authority who she trust, and the CA further delegates the right to identify other servers. Finally, some trusted CA gives a certificate to the server. The server can then prove that it is the right server trusted by those trusted third parties that the user trusts.

2.4 Decentralized Authentication and Authorization

As a motivation example, let us consider a centralized service that uses certificates for making access control decisions. The certificates are issued by a certification authority of a public key infrastructure and the service uses online validation. The service works in a similar manner to buying things with a credit card where the credit limit is checked. There are two central points in this service. The first central point is the PKI that issues certificates and the second is the validation server. The more vital for the every day use of the service is the validation service: For the globally used system, the online validation service needs to be working 24 hours a day, seven days in a week, and there are lots of validation queries. If the validation service is not reachable, nothing will work. For making the validation system more reliable, the validation system can be duplicated and the duplicates can be distributed to different locations. But this adds the need of administration and the system still has central points that at least one must works.

This background chapter has introduces systems that can be used for developing a service like the one described in the above example: X.509 or PGP certificates binds a name of a user to a public key and an access control list defines which user are allowed to use the service. Nevertheless, this traditional approach does not suite to every kinds of situation or cover all problems of trust management such as how an administrator of the service defines the access control policy.

Since the middle of 1990s, several decentralized solutions for trust management are presented. The previous sections have also described background for these other kinds of solutions that can suite better for large scale services. Instead of centralized certification authority and validation, a decentralized authorization can be used to make the service more flexible and still reliable and secure.

Eventually, the access control mechanism is one part of a trust management system. Here, access control can be considered in its wide significance mean-

ing any means to get assurance that an entity is in legal path. For access control, a trust management system describes trusted actions and trust relationships with a common language [11], i.e. both the access control policy and the credentials use the same syntax. This makes comparison of the policy, the credentials and the requested action easier.

Decentralized systems may have two kind of actors: humans and computer systems. Trust means little different thing to both of these actors: A human user is trusted if she is believed to be honest, i.e she does not have malicious intents. A system is trusted if it is believed to be secure, i.e. it opposes malicious intents directed towards it [49]. The basis of trust, the belief, should be based on knowledge.

In decentralized system, anyone can denote her believes, which makes centralized certification authorities unnecessary. The belief can be both an authentication of an entity or an authority that an entity has. For example, someone can certify the identity of Alice, or what kinds of rights Bob's program has. Of course, a belief can be deliberately or accidentally based on lies.

Often, decentralized systems use transitive trust. Because trust in human contacts is not transitive, the given trust is somehow limited. The limitation can be time, restriction of authorization, restriction based on a level of believe (e.g. certain - quite certain - probably), or a method that must be used to verify the belief from its origin.

Because trust in decentralized systems is based on believes, it is essential to check that the believes are correct. In transitive system, trust is delegated from one entity to another. In verification, the chain of trust is traced back to its first issuer. Usually this first entity is the same that receives a request, and the chain and the request form a certificate loop presented in section 2.3.1.

Some decentralized trust management systems uses a common compliance checking entity to checks that requests are acceptable. However, this does not mean that the entity is centralized, it just is for general use and every service can still have its own copy of the compliance checking entity.

2.5 The Problem Statement

The main objective of this thesis is to create evaluation method for trust management systems. The purpose is to find out what kind of authorization system is usable for users of the system, and still suite well for varying

applications. Other types of comparisons are out of scope of this work, and can be found from elsewhere. For example, Stephen Weeks has compared the mathematical background of existing trust management systems [80].

Chapter 4 introduces five different decentralized authorization systems. They are compared later in this thesis based on the evaluation criteria and comparison methodology that I present in the next chapters.

Chapter 3

Comparison Aspects

This chapter introduces three aspects that are used as basis in comparison of authorization mechanism, and describes how each aspects can be tested. The first section discusses usability and its testing methods. The main focus of this thesis is the usability evaluation of different authorization systems presented in chapter 4. Another aspect is applicability, and the second section introduces different areas where the authorization mechanism are or could be used. The third section defines scalability. The comparison methodology used in this thesis and the main focus of the comparison is presented in chapter 5.

3.1 Usability

ISO standard 9241 defines usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in specified context of use" [71]. Product can be a piece of hardware or software and, of course, the user is a person who use the product. The user should be able to reach her goals effectively (efficiency) without errors and shortcomings (effectiveness). Finally, the user should be pleased to the use of the product (satisfaction).

Jacob Nielsen specifies that usability consists of five attributes [66]. His definition is more wide than the ISO standards definition but contains all same properties that ISO definition. According to Nielsen, the usability attributes are [66]:

- **Learnability**. The user interface of a system should be so easy to learn

that users can do some work rapidly after start of using the system.

- **Efficiency**. The user can use the system efficiently.
- **Memorability**. The system should be easy to remember so that a user who has not used the system for some time can easily start using it again.
- **Errors**. If the user makes an error, she should be able to easily recover from it. There should not be many errors and none catastrophic errors that a user can accidentally do.
- **Satisfaction**. The user should be able to enjoy using the system.

These much-used and much-tested lists of usability attributes provide the backbone of usability. However, such lists of usability factors should not be regarded in any way as exhaustive, and the current discussion around usability is moving towards a more holistic approach towards usability towards an ingredient inside the broader concept of "user experience". Still, more refined elements of user experience can then build upon of the lists of usability attributes. They describe the *essentials* of usability for most systems and services.

When usability of a product is measured, the measurer need to know the users, and their goals, tasks, equipment, and the physical and social environment [41]. She should be familiar with the product and discover how effectiveness, efficiency, and satisfaction can be measured. The user works with the user interface of the product. Designers of the user interface should understand the user in addition to knowing the underlying technology. Designers must know who the user is and what she tries to do [41]. In these days when usability and user-centered design is a common practice in many application areas, within computer security and authorization mechanisms especially, usability is still taking its first steps.

In this section I describe suitable usability evaluation methods for authorization mechanism analysis but first I introduce what kinds of users an authorization system has and what kinds of tasks they perform.

3.1.1 Users

Authorization mechanisms have several user groups. For a completed product, the most obvious user is an end-user who tries to access a resource or a service. Still their primary task is not security but to use the resource or the

service. The group of end-users is growing because World Wide Web based services are becoming general. This user group consists of various kinds of people: some know the system and the underlying technology like their own pockets (expert users) while some others (casual users) know something about the system and computers, for example, how to use the mouse and that the icon that looks like a diskette usually means "save". Finally, there is a user group that does not know at all how to use a computer (novice users). However, the novice users are unlikely users of the computer authorization mechanisms because they more likely use human to human communication without computers.

Different groups of users will start using a service in different life time of the service [70]. The expert users buy any new "gadget" with any price as long as they feel the need for (at least part of) the service. A technological product must satisfy basic needs before there can be enough casual users to make the product profitable for the seller. The product must be customer-driven and human-centered before the casual users make the product becomes productive for the seller. The casual user want to perform tasks with a reasonable price and easily.

Another user group is formed by the (system) administrators, who maintain and administer the service. This other user group is the administrators. An administrator can also be a owner of the service or the resource, not just a hired employee. They should have more information about how an authorization mechanism works and how to configure it because, for example, if they do something wrong, many users suffer from misconfiguration or the whole service does not work correctly.

Last but not least user group is program designers and programmers. At first, they make the programs that other user groups use. They should know all user groups and their goals in order to make a usable service. The programmers may use an authorization mechanism to give some access rights to a program which is then used by the users. Section 3.2.3 gives an example of this case. To create usable authorization services, the programmer needs to understand also the access control mechanism that she is implementing. Probably the usability of the specifications of the authorization mechanisms some day gain attention.

Security is a chain that is only as strong as it's weakest link. The user is usually named to be this weakest link. Thus the designer of a program should know the users and her tasks and create a handy user interface to the program. The designer has also to inform the user why, for example, authentication is necessary. Otherwise the user tries to circumvent the mechanism

because she has no motivation to use the mechanism correctly [2].

3.1.2 Goals and Tasks of User Groups

The purpose of a decentralized authorization system is to proof that a user has an authorization to perform an action or use a resource. The system may not be interested in about the identity of the user. The system may even authorize software agents that act on behalf of the user. Decentralized authorization systems has different user groups presented in previous section. These groups have different goals and tasks.

A goal of a common user is to perform an action in a decentralized system. She does not particularly want to authenticate herself or proof that she has authorization to perform the action. Thus, the common user usually sees the security policy that requires authorization/authentication as necessity for the system, not as a part of her own particular task [83]. Security mechanisms themselves are so complex and hard to understand that common users have difficulties to perform security related tasks correctly [82].

Analyzing the tasks of administrators is more demanding because they need to interact with the security part of the system and perform more complicated, critical and sometimes unexpected actions. Generally speaking, the main task of administrators is to make the system works. The administrators of a decentralized system may be independent from each others and thus their tasks can vary from one part to other part of the system. For example, the system may have different administrator for common software and hardware of the system and for security policy and security credential management, as well as for different phase in the system lifetime, i.e. for installing a new node or maintaining a node. One task of the administrators is to maintain the delegation of the credentials.

The third user group, the software developers also interact with the security system of the decentralized system. Their task is to delegate credentials to the software agents they have created. They also have to define what kinds of requirements the agent has in order to work correctly and securely, i.e. what kinds of requirements the agent has for a node where it is executed.

The goals and tasks of different user groups of decentralized systems are introduced here briefly and discussed more precisely in section 5.2 where the main tasks of different user groups are introduced for the comparison of different decentralized authorization systems.

3.1.3 Usability Evaluation Methods

Different usability evaluation methods are suitable for different phases of design and implementation. If the system is ready, the user interface can be tested with a usability test and real users. The test can also be done for a similar system and find out what kind of difficulties users have in it. Standard review, cognitive walkthrough and formal evaluation are applicable in the beginning of developing process. This subsection introduces several usability testing methods that are suitable for making comparison of different authorization mechanisms.

User and Task Analysis

In user and task analysis, the developers of the system first find out who is going to use their system. Then the developers observe ordinary users in action. The developers try to find out what users' goals are, what the users do to achieve their goals, how the person (culture etc) of the user influence to the task, and how physical environment influences the user [41]. This method is suitable for the beginning of the developing procedure, for gathering the user needs requirements for product specification.

Inspection

In inspection methods, specialists who have knowledge of both technology and the users collect a list of usability problems. Jenny Preece lists several inspection methods: expert reviews/usage simulations, heuristic evaluation, walkthroughs, standard and consistency inspections [74]. For example, in standard inspection an expert inspects that the interface complies with international standards. The inspection method is useful in the beginning of the design process when the problems can still be fixed. In usability inspection methods, no real users are involved, but the inspection is performed by usability experts, often together with system designers.

Cognitive Walkthrough

Walkthrougs are on kind of inspection method. First the tester design a task from the system specification of screen mock-ups. Then the tester walk through the task and tries to figure out what the users most probably do [74]. This method can be used in the beginning of the design process and do need good knowledge of the users but not need the real users to be testers.

Cognitive walkthrough can also serve as basis for the design of a further usability testing with real users in later phases of the development process.

Formal Evaluation

Formal methods are conquering new field: usability evaluations. Researcher tries to develop a better notation for the formal specification of dialogues. They assume that better descriptions what will happen in a dialogue is necessary. If the specification is (very) good, the user interface may be automatically generated from the specification [66]. In formal evaluations, the system's usability is evaluated against design guidelines and standards. Even it is a good tool for checking e.g. consistency across a large product or product family, it does not provide any information about the real end-users of the specific product.

Usability Test

A usability test is the most important testing method in usability because it really tells how users are able to use a system [66]. The purpose of the test is either make the system better in development phase or evaluate the usability of a ready made system. In usability tests, chosen users will complete ready made test cases for a system. This sounds like an easy task but it is not. Choosing users is crucial: they represent all the users of a system and their experience of using similar systems will show in the test results. Also test cases may affect the test results, so it should be taken care when designing the tasks for the test and in choosing the users.

This section has introduced what usability means and how it can be tested. The next section discusses about applicability and gives examples of using the decentralized authorization systems.

3.2 Applicability

Decentralized authorization systems are suitable for many kinds of applications, from the network layer to the application layer. This section presents various applications based on different authorizations systems. First, middleware solutions based on authorization systems are introduced. Then I discuss about electronic commerce and other applications that uses authorization systems.

3.2.1 Network Layer Policy Management

Internet Protocol Security (IPsec)

Internet Protocol Security (IPsec) [52] provides network layer security for the Internet. IPsec offers two protocols for securing an Internet connection: *Encapsulated Security Payload (ESP)* [54] secures confidentiality and/or integrity of a datagram, and *Authentication Header (AH)* [53] ensures integrity of an Internet Protocol header and its content, i.e. the datagram. Both of these protocols work on top of the Internet Protocol (IP) securing the content of IP datagrams, i.e. the upper layer protocol datagrams such as the datagram of the Transmission Control Protocol (TCP) or the User Datagram protocol (UDP) and finally the application data packed inside those transport layer datagrams.

The use of IPsec is transparent for applications and their users, but it needs a security policy for connections. A security policy defines what kinds of connections are allowed. It gives, for example, security methods, and authentication and encryption algorithms that can be used with ESP and AH. IPsec defines that possible policies are stored in a *Security Policy Database (SPD)*, but it not describe how this database should be implemented [52].

Similarly, IPsec defines another database, a *Security Association Database (SAD)* for storing the security parameters of the ongoing connections. The agreed set of security parameters used in one way in a connection is called a *security association (SA)*. The header of IPsec AH or ESP datagram contains only an index number for the security association that defines how this connection is protected. The correct security association is found from the SAD database and the datagram can be unpacked based on the SA information.

IPsec proposes *Internet Key Exchange (IKE)* [43] protocol for creating security associations needed in Virtual Private Network (VPN) connections. IKE is a two phase protocol. In the first phase, a security association for secure forthcoming negotiations is established between two network nodes. Then, in the second phase the actual security association for protecting the communication of the specific purpose protocol is established. The idea is, that between two communicating parties, there may be several connections protected in different ways.

Trust Management for IPsec

Matt Blaze, John Ioannidis and Angelos Keromytis suggested the use of a trust management system for managing network layer security (IPsec) protocols in 1999 [15]. They argue that the trust management architecture must be efficient because from every incoming and every outgoing packets must be examined that what kind of protection they need. Their other claim is that for common purpose network layer security the language used to describe policy must be expressive, so that every kind of situation can be handled.

The problem of network layer policy management can be divided to two: the first is to establish a new security association for a new connection (trust management) and the second is the handling of every incoming and outgoing packet (packet filtering) [14]. The creation of new security association happens rarely compared to the handling of incoming and outgoing packets. The creation of SA can use more expressive method than a common packet filter needed for handling efficiently the packets [42].

SA establishment with trust management system raises two now problems: how to discover all the needed credentials and what capabilities the credentials give to an entity [14]. The first problem is solved with a simple protocol that ask from the peer, the responder, all the credentials issued to the other peer, the initiator. The responder can also give hints about servers that may contain more credentials. When all credentials is collected, the trust management system of the initiator can check what kinds of connections are allowed between the initiator and the responder. Finding out in advance what capabilities the initiator has, gives change to avoid unacceptable propositions of security associations.

Matt Blaze, John Ioannidis and Angelos Keromytis has published an Internet draft about the compliance checking and IPsec policy management [13] but the draft is expired. Niklas Hallqvist and Angelos Keromytis have implemented an IKE protocol that uses KeyNote trust management system for the phase two negotiation [42]. The phase one negotiation is simple and using of complex trust management is not necessary. When a new security association for a connection (in the phase two negotiation) is established, first the SPD database is checked for need of protection. If the connection needs protection, a negotiation with the communicating peer is initiated with IKE. KeyNote credentials are used in the negotiation for denoting what kinds of security associations can be used. When the security association for the connection is established, the databases are updated: The needed credentials are stored in a credential database, used security association is stored in the SAD database, and the SPD database is updated that packets of this connection

uses IPsec.

3.2.2 Secure Middleware

CORBA

Common Object Request Broker Architecture (CORBA) [40] is developed to make distributed object-oriented infrastructures easy to implement in a heterogeneous environment. CORBA works as transparent middleware between clients in different kinds of environments and resources in different kinds of environments, too. It hides from the client objects where the target objects are and how the objects are implemented. In the CORBA architecture, objects communicate with each others through Object Request Brokers (ORBs) and operations offered by objects are described with Interface Definition Language (IDL) that is independent from programming languages.

CORBA offers basic services such as search of services based on their properties or names, informing about events, methods for several objects to participate a transaction and using of concurrent services. CORBA provides also security services offered by ORB, i.e. the service is almost invisible for the object. When a secure ORB is used, the client object first identify itself to the ORB. After identification, the client object can contact to services and the ORB will automatically provide credentials of the client for the services access control, and the connection is secured according to a security policy.

Even though CORBA specifies several security services, they are not usually used [56]. Many CORBA products have not implemented the security services, or use only the network traffic encryption. Applications may deliver the credentials directly to a centralized service that makes the access control decision. But centralized access control method is not scalable, and thus something else is needed. As an example, the next subsection describes credential based access control method that uses CORBA security services and authorization certificates.

Authorization in CORBA

In his master's thesis project, Tuomo Lampinen has implemented a certificate based authorization in CORBA [57]. Purpose of the work was to solve a situation given in the figure 3.1. A service provider (e.g. Alice) delegates selling of his service to a retailer (e.g. Bob), and a customer (e.g. Carol) buys from the retailer the access rights for the server of the service provider.

Authorization certificates are used as credentials to prove the access rights of the customer, and also for identifying the retailer to the service provider. CORBA security services provides the transparent delivery of credentials between customers and servers and making the access control decisions.

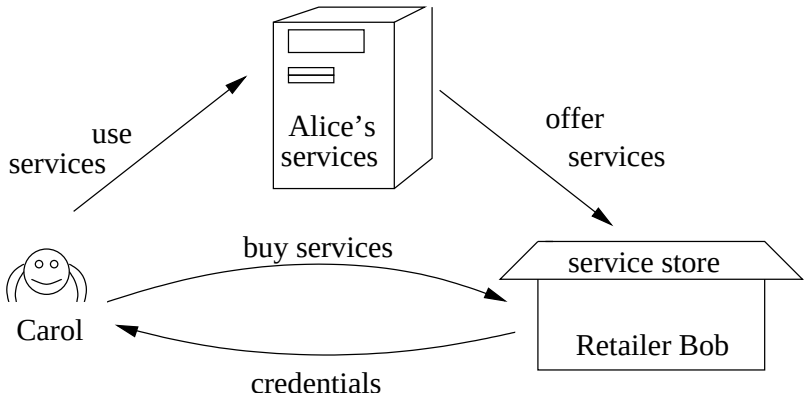


Figure 3.1: Authorization in CORBA [57]

First, the service provider Alice gives for the retailer Bob the right to resell her service by issuing an authorization certificate. This certificate gives right to delegate the access rights of the service. When the customer Carol needs a service, she first has to find a retailer who sells the service. This can be done, for example, with the trader service of CORBA. The trader service checks the given description of a service, e.g. a service store, and it gives the location of the service to the customer if it knows such a service. Then, Carol can contact to the retailer and buy access right for a service she wants to use. She gets a certificate that denotes the right she has bought. The certificate may be valid, for example, for one month or for one turn.

The security service of CORBA is used to pass the authorization certificates from the customer to the server. Carol form a connection to her ORB broker and gives the credentials she has bought to the broker. CORBA offers a challenge response service for verifying that the requester really have the corresponding secret key of the public key stored in the certificate. The ORB broker delivers transparently the credentials and verification result to the service provider when Carol uses the service.

In the service side, the access control mechanism of CORBA verifies that the certificates give authorization to perform the requested action, i.e. the certificates form a valid chain. All credentials may not be in the request, and then the service must fetch missing certificates from directory services.

Tuomo Lampinen gives several advantages of using authorization certificates for access control in CORBA based applications [56, 57]. The use of authorization certificates in CORBA access control makes the access control policy transparent to the application developers. The customer object can be anonymous and still prove that it has the right to use a service. The authorization to do something may be derived from several sources and the authorization can be delegated further.

In CORBA based applications, services in unknown location are used transparently probably through a network. In the next subsection, I give an example of how a program found from a network can be safely used in an environment.

3.2.3 Policy Management for Mobile Code

Mobile code is a piece of code that is received from a potentially untrusted place and is requested to be executed in a host [9]. Because using of unknown code in a host may, for example, infect a virus to the host or create a back door for a cracker, some method for access control of pieces of code must be created. These threats are not new but the mobile code has made the scale of the problems larger [37].

One attempt to create secure mobile code is based on the signing the programs. But, signing the code is not enough [37]: The host should somehow be able to know if the signer can be trusted, i.e. know the signer of the code. Other problem is that signature just gives "on off" property for the program. It cannot tell what kinds of execution rights are enough for executing the piece of code in this particular host.

Matt Blaze *et al.* [9] gives two roles for trust management in mobile code security: expressing trust relationships between entities involved and denoting the minimal set of capabilities that is needed for the execution of the program. Jonna Partanen has implemented such a system for Java 2 policy management [72].

Java 2 Security Architecture

Java offers operating system independent programming language and Java applets have made the mobile code widely used. A Java program can be packed in a container called jar-file, and that jar-file can be signed and distributed through network. Java access control divides code to trusted and untrusted code based on the signer of the code and a location where the code

is fetched. Security policy of Java environment is written in a configuration file, and it denotes which signing keys are trusted and what kinds of access rights are given to the code.

Based on the security policy, a piece of mobile code is placed in a protection domain. The protection domains gives certain permissions to all the objects that belongs to it. A permission gives the right to use a protected resource. When the piece of code is executed, it have only the permissions given to the protection domain that it belongs to. Java programs cannot extend their permission by calling another programs that have more rights. When a program calls another program and this program tries to use a protected resource, the access control checks also the permission of the calling program.

Distributed Policy Management for Java 2

Access control in Java 2 is based on a configuration file that contains allowed signers of program code and permissions given to the signed code. This kind of solution does not scale well to work on a large network [68]. Instead of centralized access control, authorization certificates can be used to provide distributed policy management for Java code. The distributed policy management for Java 2 was briefly introduced in an example above in section 2.2.1. In this subsection, I give more detailed description of the solution.

For providing distributed policy management with authorization certificates, a new subclass must be added to the general Java certificate interface. The current Java 2 provides only an interface for X.509 identity certificates. Jonna Partanen and Pekka Nikander have implemented SPKI authorization certificates for Java [73]. A new certificate has two parts, the certificate and its signature. The certificate consists of all the fields of the SPKI authorization certificate. The SPKI certificates are introduced later in section 4.4 in details.

The Java policy is changed to support authorization certificates instead of configuration file. In the new system, the Java permissions are presented as authorization certificates. The protection domains has set of certificates presenting the permissions it can give to code. In standard Java, the protection domains are static and cannot be modified after creation. The policy based on authorization gives also a possibility to dynamically change the permissions of a protection domain.

When a piece of mobile code stored in a jar-file requests to be executed, the authorization certificates stored to the file are given to policy to be evaluated.

Each program get its own protection domain which permissions are given in the certificates. Of course, the certificates must be verified first. The policy firstly checks that all the certificates are valid, i.e. signatures are correct, the chain is intact from the first issuer to the last subject and that the delegation is allowed by all but the first certificate. The first certificate of the chain must be issued by the policy itself, like in figure 2.6, because the policy decides who are authorized to use what protected resources. If everything is integral, the chain is valid and ends up to the piece of mobile code, the code has the permissions to use the protected resources given in the certificates.

SPKI authorization certificate based distributed policy management can be used to solve both the problems of mobile code security given in the beginning of this subsection. The certificate can describe precisely what permission the code needs in order to work. Because of delegation, the certificates can form chain that denote trust delegated from one entity to another and finally to this code [68].

Decentralized Trust Management for Jini

Jini is Java based networking technology that provides framework for building distributed applications [6]. The framework provides lookup service. The service providers register their services to the lookup service for certain time period. When a client want to use a service, she asks from the lookup service who offers the service she needs. She gets a proxy that will provide connection to the actual service.

In his Master's Thesis, Pasi Eronen has extended the Jini to use decentralized trust management [32]. In his implementation, trust relationships between the entities of the Jini architecture are presented with SPKI authorization certificates. The entities are Java programs that can use SPKI certificates for access control management as presented earlier in this section.

3.2.4 Electronic Payment Systems

Many systems have been proposed to provide secure electronic commerce and banking but none have spread to world wide use. Nowadays electronic commerce often means that a user order something from a WWW shop and pay it with her credit card like any other merchandise. More problematic situation occurs if the merchandise has little value, i.e. the user wants to pay a purchase with a cash-like system because currently used systems are too heavy for micropayment. Decentralized authorization systems can provide

solutions to electronic payment in small scale.

Matt Blaze *et al.* has introduced a micropayment system [16]. The participants of the system are presented in the figure 3.2. A merchant has something to sell that a payer wants. Both signs up to a Provisioning Agent (PA). The payer can use, for example, mobile phone or other personal digital assistants (PDA), and the merchant uses Merchant Payment Processor. The system contains also Clearing and Settlement Center (CSC) for conciliation in conflict situations.

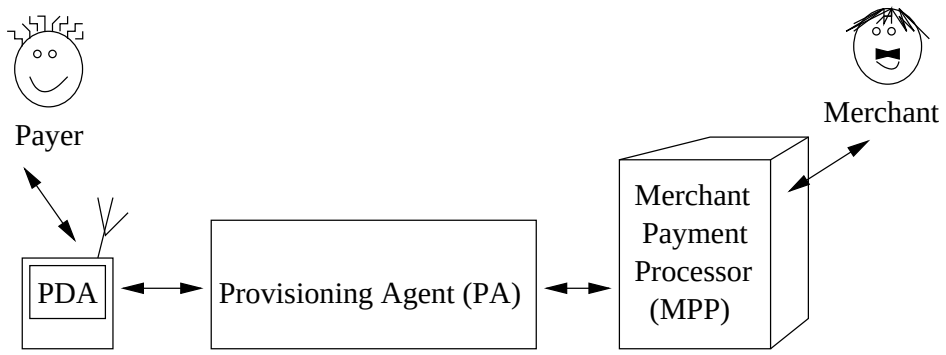


Figure 3.2: Micropayment architecture

In the trust management based micropayment system, the merchant issues authorization certificates that defines her security policy. The security policy defines trusted Provisioning Agents that act as trusted third parties for the merchant. PA delegates the right to use an shop application of the merchant to payers. Each payer may have different kind of right to the shop.

When the payer finds something to buy from the shop of the merchant, the merchant will generate an offer for the purchase and sends it to the payer. The payer then creates a microcheck, i.e. a certificate, based on the information given in the offer. The changing of the offer is not possible because the merchant uses statefull system which remembers the offers for a certain time.

The three certificates, issued by the merchant, PA, and the payer, will form a certificate loop. If all certificates are valid, then the merchant can trust that the check of the payer is valid and she can send the purchase to the payer.

The authorization certificate based micropayment system presented here is simple. It allows to define what kinds of risks are possible to take for a merchant.

This section has introduced several kinds of fields where decentralized access control management can be used. In the next section, I discuss about scalability and its problems. Even though scalability is out of scope of this thesis, the next section shows one aspect why decentralized authorization systems are better than identification based systems.

3.3 Scalability

Clifford Neuman argues that scale in distributed system has three dimensions: number of users and objects (numerical dimension), distance between parts of the system (geographical dimension), and amount of administrate work that the system needs to work correctly (administrative dimension) [64]. He defines that a system is scalable if the number of users and/or objects and distance between parts of the system can increase without a significant increase in amount of administrative work or losses in performance. Distribution of a system gives more flexibility for creating more scalable systems but it also brings forth new problems that can increase the amount of administrative work.

Testing of scalability is out of scope of this thesis. Still, I want to describe what scalability means and what kinds of problems scale causes for systems. Scalability is one good reason for changing from a centralized system to a decentralized system, and from identification and authentication to authorization of the acting parties of the system.

3.3.1 Dimensions of Scalability

Let us consider the numerical dimension of scalability from the first point of view. A larger number of users will require more capacity from a service or otherwise its load will increase beyond the usable limits. One way to reduce the load is to build a distributed system. Other solutions include replication of the service and caching [64]. Actually, replication and distribution are quite similar, and caching can be considered as a form of replication. The difference between replication and distribution is that when building of a distributed system, the needs of users "near by" can be taken better into account instead of just replicating the system.

Other point of view to the numerical dimension is the number of organizations that participate to a system. If the system grows from one organization to two or even more organizations, the amount of administrative work will increase.

In a large decentralized system, no single entity is responsible for management of the system. The organizations may have, for example, different security policies which may need to be somehow unified that the service is able to define legitime users. The decision making will take more time because every decision have to be discussed in the participating organizations and also together.

When the amount of users increases, usually also the amount of computers and other resources increase. If the service is decentralized and used in several organizations, these computer may vary from their operating systems and application layer solutions. This will lead to increase in the administrative work load.

The geographical dimension of scalability must be considered when a system is distributed to several locations. The communication between the parts must taken care of in a efficient and secure manner. The network solutions will effect to the communication by causing delay and unreliability from both reachability and security points of view. Remote resources may leave more tasks to the user part of a system. This reduces communication between parts and load on the remote resource but the clients become more complex and need the more local information [64].

3.3.2 The Problems of Scale

Among the largest problems of decentralized systems are naming and security. The problems can be seen in the X.509 public key infrastructure. Simply, human names do not scale to globally unique names, and centralized but hierarchical structure of certification authorities issuing of certificates, i.e. evidence of trust, do not provide global trust. In this subsection, I discuss about problems of scalable naming, security, and trust.

Naming

As mentioned earlier in subsection 2.2.3, human names work well for identifying members of a family. When the group of users grows up, the unique identifiers will be problematic. Human names are not the only problem, people use names also for files and computers.

As described earlier in section 2.2.3, hierarchical solution is used to solve the problem of scalable naming. However, hierarchical naming does not solve global naming, but may offer more scalable solution for the problem.

Security

Security of a large and versatile system is hard to organize because there are more possible vulnerabilities [64]. A decentralized service administered by several organizations may work upon different kinds of underlying technologies. Unified mechanisms can be hard to find but may not be needed either. For example, if one security solution of one part of the decentralized system fails, the system can still work securely based on the other parts.

Compared to a centralized system, decentralized systems are also more complex from the security point of view. It is not enough that a node is sure that a user of it has authorization to use resources offered by the node. The user or an agent acting on behalf of the user need to sure that the node is trustworthy and complete the given task securely and honestly. Both the user (or her agent) and the node can try to misuse the system either by accident or on purpose which must be taken into account when establishing a security for scalable decentralized systems.

Security systems express trust between entities of a system. For example, an access control list gives users that can use resources of the system and are trusted to be honest, i.e. even than they could misuse the system the do not do so. The next subsection discusses about trust in scalable systems.

Trust

Trust between humans does not scale large amount of people because human trust is not transitive [1]. Contradictory, global certification authorities are based on transitive trust. The certificate authorities form a hierarchical tree with one root CA. CAs can also be seen as a forest with several trees and several root CAs that trust each others. A user gets her certificate from some lower layer CA. When the users interact, they have to trust all the CAs of the system.

The hierarchy and transitivity of trust is not the only problem in scalable systems. The root of a CA tree must be trustworthy. For example, some scepticism could be point at towards the currently used X.509 based WWW server certification. The most trustworthy CA is considered to be VeriSign. VeriSign is not a governmental organization, it is a company listed in the Nasdaq stock exchange. Nowadays, everybody is expected to trust VeriSign. But can a company whose purpose eventually is to produce money to its owners be trustworthy? For example, many has trusted to Enron that was a large electricity company. Surprisingly, Enron went bankrupt in December

2001 and many lost their life savings but some managers sell their stocks before the value of stock collapsed.

It is not enough to trust to the CAs. Users must also trust to the implementations of programs they use. In a decentralized system, there may be several software developers. Let us consider commonly used WWW browsers as an example. They lists several dozens of trusted CAs. The implementations of WWW browsers are far from perfect. For example, in August 2002 a bug was found from Internet Explorer (a popular WWW browser). The bug allows anyone who has a valid VeriSign certificate for her WWW site to forge any other VeriSign site certificate because the browser does not verify the certificate chains [39].

As discussed earlier in the previous chapter (in sections 2.2.3 and 2.4), trust does not scale well. One solution to provide trust for larger system is to divide the system to "partial networks". Each parts is trusted differently and more fine classification of trust is used for entities of the same "network".

3.3.3 Users and Scalability

Scale influence to users as well as to a system but how the scale effects to the user is not much researched [64]. For example, users must be able to handle naming and security issues. Users should have a mental model of the system and, in some level, understand how the system works [74]. Otherwise, the using of system cause problems when user does not understand why something happened. For example, the underlying network in a decentralized system should not be hidden from the user because it effects lot to the functioning of the network. How much to hide and what to show to the user for the system to remain as simple as possible, and yet to be understandable and informative enough to enable the user to build a mental model of the basic structure and logic behind the system, is a difficult question, to which there is no easy answer [82].

Even than the testing of scalability is beyond the scope of this thesis, I want to note that authorization systems are as such more scalable than authentication systems. They do not need centralized authorities, and there may be several location for needed information and resources. In a decentralized system there can even be several servers offering same kind of service with same conditions. However, trust may become problematic in a very large scale system where users and parts of the system does not know each others.

Chapter 4

Different Approaches

In this thesis, I have concentrated on five different approaches of decentralized authorization. The first trust management system was PolicyMaker [11, 12]. KeyNote was developed according the same principles as the PolicyMaker system [8]. Both of these are partly work of same people (Matt Blaze and Joan Feigenbaum). The IETF Simple Public Key Infrastructure (SPKI) working group has developed a (simple) certificate structure and operating procedures for Internet trust management [29, 27]. At the same time, in Massachusetts Institute of Technology the Simple Distributed Security Infrastructure (SDSI) was developed to be a simple public key infrastructure that provides groups that have clear terminology for design access-control lists and security policies [77]. Development of SPKI and SDSI has been combined together in 1997 [22]. The Telecommunications Software Security Architecture (TeSSA) is an infrastructure for trust and policy management created at Helsinki University of Technology [67]. It is based on SPKI certificates among other work of IETF Security Area development. This chapter introduces all the above mentioned authorization systems.

4.1 PolicyMaker

Matt Blaze, Joan Feigenbaum and Jack Lacy introduced the *PolicyMaker* Trust Management System in their paper in IEEE's Symposium on Security and Privacy in the year 1996 [11]. It was the first tool for processing signed requests and check if they comply with the policy [9]. Existing services for trust management were PGP and X.509 which left open the question how to express the security policy and how to use the information gained

from the certificates. The PolicyMaker system consists of a language for specifying policy and trust relationships and a mechanism for checking that a request complies with the policy. This section is mostly based on PolicyMaker publications [11, 12].

4.1.1 General Principles for Trust Management System

Blaze *et al.* [11] noticed that the trust management problem has not previously been studied in its own wholeness. They identified the problem and introduced a solution that is independent of any particular application. The trust management consists of security policies, security credentials and trust relationships. For example, in a bank a policy says that at least three bank officers need to accept a loan of million dollars. A bank officer needs a credential that show that she can be counted as one of those approved officers. In the bank someone can issue such credentials, i.e. there is a trust relationship between the bank and that officer [11]. The result of Blaze *et al.* work, the PolicyMaker system is based on the following general principles [11]:

- **Unified mechanism** (common language). Policies, credentials and trust relationships are expressed with a same language. An application can handle security in a way that is comprehensive, consistent and almost transparent.
- **Flexibility**. An application can use the same system to express both simple and standard policies, and complex trust relationships in very large-scale.
- **Locality of control**. Each party can decide for itself which credentials and third parties are trustworthy. Global hierarchy of certifying authorities is not needed.
- **Separation of mechanism from policy** (general compliance-checking mechanism). All applications can use the same certificate verification infrastructure because the system does not depend on the semantics of the applications.

4.1.2 Architecture of PolicyMaker

PolicyMaker acts like a database query engine. An application sends a request, local policy statements of the application and a collection of credentials

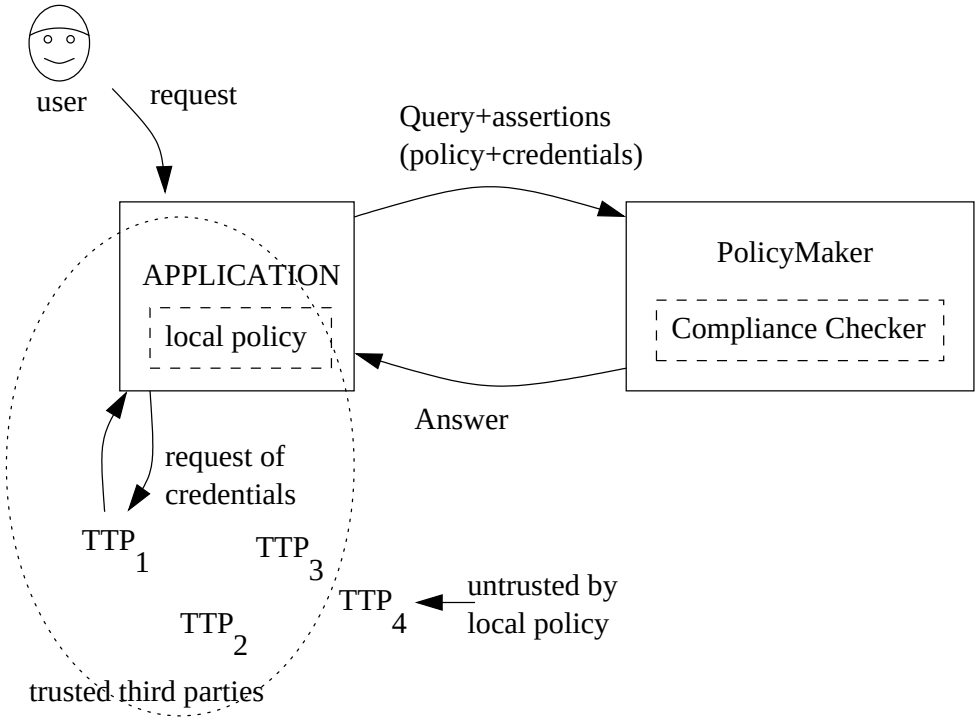


Figure 4.1: PolicyMaker

to PolicyMaker. All of these are expressed in a simple language provided by PolicyMaker. Then the compliance checking algorithm of PolicyMaker verifies if the credentials form a proof that the request complies with the policy of the application. It returns a simple yes or no answer or additional restrictions that would make the action acceptable [11, 12]. Figure 4.1 show how an application sends a query to PolicyMaker.

First, the application collects all needed credentials. All security information is not usually stored in one local place, and thus, the information must be gathered up from different trusted third parties. The local policy can define to which trusted third parties it trusts and it can limit the area of trust of these third parties. A trusted third party itself can also gather information from other. Like said above, the local policy may set a limit to how many steps can be taken. For example, a trusted third party defined in the policy can ask from other third parties and their credentials are accepted, but credentials of a query where the another third party further asks from other third parties are not accepted. In PolicyMaker, trust is monotonic. It means that each

statement can only increase the capabilities.

When all necessary security information is collected, all credentials are verified: that they are unrevoked, the signatures are correct, and they are granted by allowed trusted parties. Because the verification of certificates is the task of the application or an external agent, PolicyMaker is not dependent on a specific signature scheme. For example, PGP can be used.

But why a common trust management system, like PolicyMaker, is necessary even though it left much responsibilities to the applications? Why an own policy compliance checker for every application is not a good solution? Blaze *et al.* give several advantages of a general compliance checker [12]:

- Design and implementation of the compliance checker is sound and reliable.
- Needs of the application are not underestimated because all kind of situation is evaluated.
- The answer depends only on the input, not implicit policy decisions or bugs.
- Complexity of conditional delegation is not underestimated.
- Adding new properties to an application is easy and the compliance checker does not need any changes.
- Interoperability is easy because requests, credentials and policies are written in the same standard format.

In the next section, I explain what kind of language is used in the queries and how the credentials form a chain of trust that gives to the application assurance that the chain is valid. Then the compliance checking algorithm is presented.

4.1.3 The PolicyMaker Credentials and Queries

Trust information is contained in *assertions* in PolicyMaker. There are two kinds of assertions: *certificates* (signed assertions) and *policies* (policy assertions). Policy assertions are not signed because they are local i.e. they are defined by the application that makes the query. Security policy assertions forms a "trust root" of the application which is used when the signed assertions are considered. Both types of assertions are same form:

Source **ASSERTS** *AuthorityStruct* **WHERE** *Filter*

The *source* of the assertion is the public key of a third party if the assertion is a certificate, or the local policy in a policy assertion. The assertion binds a public key or a sequence of public keys, called an *authority structure*, to a predicate, called a *filter*. An authority structure can also be more complex structure than just a bunch of keys. We can think the above example where three bank officers need to accept a loan. In an assertion, the authority structure is then "at least three of the following keys of bank officers".

A filter is the claim that action strings in the query must satisfy. Otherwise the assertion does not hold. Filters define credentials and security policies and bind them together with public keys. PolicyMaker supports three filter languages: a regular expression system, an internally developed safe version of AWK (called AWKWARD) and a macro language of safe AWK. Java can also be easily added [11]. More about these filter languages is later in this section.

An application sends queries to the PolicyMaker system in order to find out if a public key (or group of public keys) is permitted to perform an action according to a policy. The form of a query is

$key_1, key_2, ..., key_n$ **REQUEST** *ActionString*

An action string is an application specific message. The PolicyMaker system does not understand the action strings, only the calling application can generate and interpret them. The action string can admit many kinds of capabilities, for example signing a contract on behalf of a company.

The assertions and the queries are linked together. The assertion source trust that the keys in the authority structure are authorized to perform the action specified in an action string i.e. pass the filter. First, a policy assertion binds an authority structure to a filter. For example, a bank policy says that there must be three signers in a contract giving a loan of s million dollars. (The assertions and request of this example do not obey the PolicyMaker syntax. Examples with the correct syntax are given later.)

Policy of the bank **ASSERTS** *three keys from the group of bank officers keys* **WHERE** *sign a loan contract of million dollars*

When someone asks for this loan of million dollars, a query with keys of bank officers is passed to PolicyMaker to be evaluated:

$Key_1, Key_2, and Key_3$ **REQUEST** *sign a loan contract of one million dollars*

PolicyMaker now compare the requested action string (*sign a loan contract of one million dollars*) to the filters in the given credentials. It does not know the significance of the question but it can find the policy assertion and see that three keys are needed. Correctly, the request contains three keys but based on the only given policy credential, the PolicyMaker compliance checker cannot say if the keys are valid for the action. Another assertion is needed:

Policy of the bank **ASSERTS** $Key_1, Key_2, Key_3, ...Key_N$ **WHERE** *group of bank officers keys*

Now the compliance checker can give a positive answer to the query because the keys given in the request fulfills the filters.

Like in the above example, a query usually consists of several policy assertions and certificates (signed assertions). In the most simple case, the authority structures and the query contains only single keys. Assertions can be illustrated as a directed graph. Vertices of the graph are the keys or policy sources and edges are filters. In figure 4.2 is an example of a simple assertion and corresponding directed graph. In the example, v is a source of a policy assertion that gives to an authority structure w rights that fulfill the filter f.

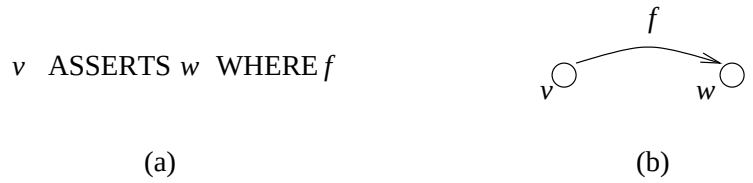


Figure 4.2: (a) An example of an assertion (b) Corresponding directed graph

The query complies with the policy if the directed graph has a *chain* $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_t$ where v_1 is a local policy source and v_t is the key in the query. A more complex query can also be described as a directed graph but then the vertices are authority structures and the v_t is an authority structure where the keys of the query are acceptable input.

There are two form of filters. The simplest filter only accepts or rejects the request. In this case, a directed graph contains a chain where all the

filters accepts the action string of the query. Other type of filter can add restriction *annotations* to action string that otherwise is acceptable. These annotations are restrictions that are not in the query. Assertions with this type of filter have actually two filters: one normal kind of filter predicate and one annotation.

If an application uses the two filter mode to check assertions, the query is evaluated in two phases. First, the validity of the action string is examined in all annotations from policy to query. All annotations are added to the action string. Second phase is executed if all annotations are fulfilled. The chain is gone through again with action string that contains all the additional annotations. In this phase, the predicate of the filter can only reject or accept the string. The two phase processing of actions string guarantee that all annotations previously added to action string are accepted in all nodes of the chain. If the query is approved, the action string with annotations are returned to the application.

Filter Languages

Like mentioned earlier in this section, PolicyMaker supports several languages for filter definitions. The language must be "safe" i.e. it can be interpreted correctly. The filter language is easier to design than a general purpose programming language [11]. This subsection introduces shortly two filter definition languages: regular expressions and AWKWARD.

A regular expression is a string built up recursively from simpler regular expressions by certain rules that are [4]:

- There is an **empty string** in the language, denoted with epsilon ϵ.
- If a is **a symbol** of the language, then it is a regular expression that represents a language that consists the string "a".
- **Parenthesis** can be used to group strings and to give them different precedence order.
- **Repetition** is denoted with Kleene star: a regular expression a^* means that the language contains strings where is zero or more of symbols a. (Markings a^+ means that there is one or more a)
- **Concatenation** of two regular expressions $a \cdot b$ gives a language containing a string "ab".

- **Alternation** of two regular expressions $a|b$ denotes a language where is two strings: "a" and "b".

Regular expressions are used, for example, in compilers while recognizing words of a programming language. They are used similarly in the PolicyMaker compliance checker to recognize symbols. For example, a public PGP key that is encoded in hexadecimal in an assertion is for of a regular expression:

$$\text{PGP: } (0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)^*$$

AWKWARD programming language developed by Blaze *et al.* is a safe version of AWK programming language. AWK [3] is a simple programming language where programs are sequences of pattern - action(s) pairs. When a program is executed, the input is compared to the pattern part of the program. If it matches, the action(s) is executed. AWK has only two basic datatypes, numeric and string, and few statements, such as if, while, and for. Regular expressions are often used as pattern test.

PolicyMaker Syntax

The appendix in [11] gives grammatical rules of PolicyMaker assertions and queries. As we can see, the syntax is really simple because the PolicyMaker do not actually understand the content of queries. UPPERCASE letters are used for terminals that are the basic symbols in the assertion or query string. Before a colon is a term that is defined after the colon. A pipe (character |) gives an alternative definition to a term. Like mentioned above, there is two kinds of assertions, simple and complex assertions:

```
assertion  :  source ASSERTS authstruct SEMICOLON
           |  source ASSERTS authstruct
              WHERE filterlist SEMICOLON
```

The source of an assertion is either a policy or a keyid. The authority structure is either a filter program or a keyid.

```
source  :  POLICY
        |  keyid

authstruct  :  filterprog
            |  keyid
```

The keyid first gives an identifier of a used key and then the key itself after a colon.

```
keyid  :  system COLON string
```

Now we can give examples of a keyid and a simple assertion where one key announces another key for some purpose that the application may know:

example of keyid:

```
PGP:''0xabcdefabcdefabcdefabcdefabcdef''
```

example of assertion:

```
PGP:''0xabcdefabcdefabcdefabcdefabcdef''
ASSERTS
PGP:''0x01234567012345670123456701234567012345'';
```

In a complex assertion, a structure for a filter list, a filter name, a filter program and a language and are needed. The filtername tells what language is used for the filter. A PREDICATE is simple filter expressed with regexps. An ANNOTATION filter consists of a simple AWKWARD program.

```
filterlist  :  filtername EQUALS filterprog
            |  filtername EQUALS
               filterprog COMMA filterlist

filtername  :  ANNOTATOR
            |  PREDICATE
            |  COMMENTARY
            |  APPLICATION

filterprog  :  language COLON string

language  :  AWKWARD
          |  REGEXP
```

In addition to the assertions, PolicyMaker needs a query. It uses same kind of syntax than the assertions:

```
query  :  keylist REQUESTS
          string condition SEMICOLON
```

```
where
keylist     :  keyid
            |  keylist COMMA keyid

condition   :  <nullstring>
            |  WHERE filterprot
```

4.1.4 PolicyMaker Example

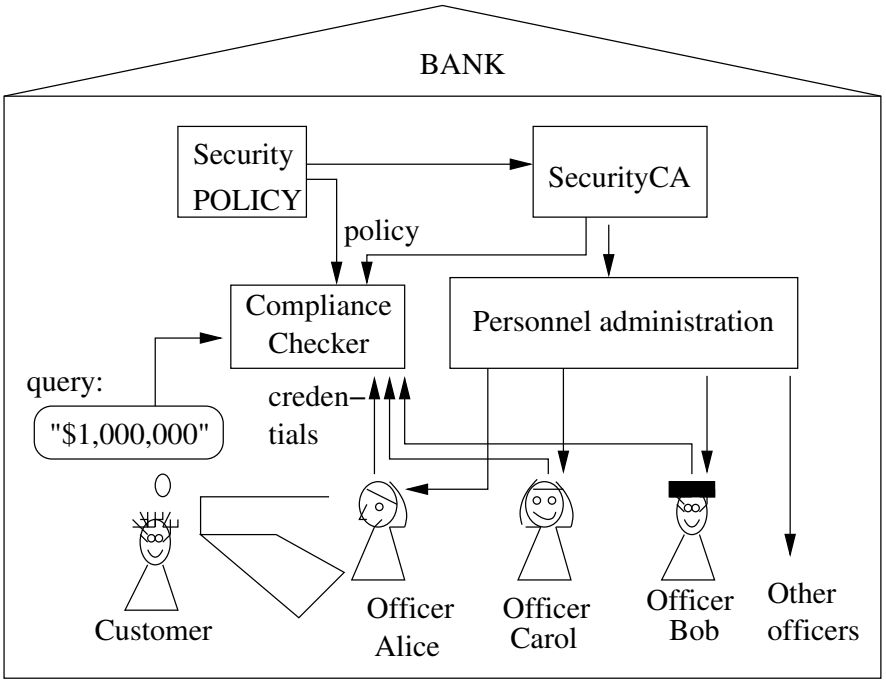


Figure 4.3: An example of bank loan

Let us form now a real query for the above bank example where three bank officers were needed to sign a loan contract of one million dollars [11]. The chain that is formed from the assertions and query is shown in figure 4.3. A loan automate first collects and verifies all needed assertions. First assertion is a policy assertion giving a name to a security certification authority of the bank:

```
POLICY ASSERTS SecurityCA WHERE PREDICATE=regexp:
        "Organizations:  Bank of Alice"
```

An another assertion issued by the security certification authority defines that three bank officers are needed to for giving a loan of one million dollars.

```
SecurityCA ASSERTS "an AWKWARD program requiring at least three
  keys directly certified by PersonnelKey" WHERE PREDICATE="an
 AWKWARD program that checks that amount is one million dollars
```

When this example was presented above, it was simplified. Now the example is more accurate. We can already to see from the above assertion that there needs to be someone (a PersonnelKey) that has right to authorize the keys of the bank officers with following assertions:

PersonnelKey ASSERTS Key_A WHERE PREDICATE=regexp:"Signer's name is Alice

PersonnelKey ASSERTS Key_B WHERE PREDICATE=regexp:"Signer's name is Bob

PersonnelKey ASSERTS Key_C WHERE PREDICATE=regexp:"Signer's name is Carol

Of course, there are other keys issued to other bank officers as well but in this example, we need only three of them in the following query:

Key_A, Key_B, Key_C REQUESTS "loan of million dollars
Organization: Bank of Alice
Signer's name: Alice
Signer's name: Bob
Signer's name: Carol

This subsection has shown the first tasks of an application: the forming of queries and assertions. Then the application checks that the credentials are not forgeries, i.e. it verifies the signatures in the credentials. The application decides how the verifications are done, the PolicyMaker system does not define any way itself. When all is ready, the application sends its policy assertions, signed assertions (credentials) and the query to PolicyMaker. The core of the PolicyMaker system is the compliance checking algorithm that verifies if the given credentials is the needed proof that the query complies with the policy. The reasons why a trust management system should use one common compliance checker is discussed in section 4.1.2 above. The next subsection discusses about the compliance checker of PolicyMaker.

4.1.5 Compliance Checker of PolicyMaker

The compliance checker and its algorithm are defined in [12]. First in the paper, a problem of Proof of Compliance (POC) is discussed. Because POC is a very complex problem and in the most general form of it is undecidable, Blaze *et al.* gives some limitations to make a useful version of compliance checking algorithm [12]:

- **Global runtime bound** give a upper bound for the time that is used for solving a problem. The bound is $O(N^d)$ where N is the length in bits used to code the input (i.e. the query itself and all the assertions) and d the given bound.
- **Local runtime bound** limits the time used to handle each assertion to be $O(N^c)$ similarly than the global runtime bound. c is the given bound and N is the length in bits of the assertion. This limitation is necessary because an assertion may create new annotations which makes the assertion longer than the original query was.
- **Bounded number of assertions in a proof** gives the maximum number of assertions that can be used in the query validation.
- **Bounded output set** limits the size of an acceptance record and the number of action strings. Both the acceptance record and the action string are explained later in this section.
- **Monotonicity** means that an assertion approving something will also approve a subset of that something.

Notation

The Compliance Checker of PolicyMaker has its own notation. *Assertions* are pairs of a function and a source. A policy assertion is marked with (f_0, POLICY) where f_0 is a function written with a programming language. Similarly, credentials are marked (f_i, s_i) where f_i is a function and s_i is a string presenting a source ID that can be a name of a person or a company, a public key etc. Like said earlier, the compliance checker do not actually understand the content of the queries or assertions, for example, the source ID is just a string.

Assertions can be divided to *acceptance records*. Each assertion(f_i, s_i) can create several acceptance records (i, s_i, R_{ij}) where i is the assertion number,

s_i the source ID, and R_{ij} an *action string*. The action string R_{ij} can be a conditional version of string R or it can be same as it. The string R corresponds the query r.

Matt Blaze *et al.* [12] gives an example of a conditional approval of a loan: "a loan can be given if both Alice and Bob will accept it". In this example, the credentials are (f_1, A) and (f_2, B). Both functions are "I approve the loan if and only if the other approves it", i.e.

$$\text{(A approves)} \Leftrightarrow \text{(B approves)}.$$

When compliance checker starts evaluating the query r, it starts from an *initial acceptance set* that contains only the acceptance record of the searched query string, i.e. $\{(\Lambda, \Lambda, R)\}$. Then all the assertions delivered with the query, the policy (f_0, POLICY) and the credentials from (f_1, s_1) to (f_{n-1}, s_{n-1}), are evaluated to be acceptance records. This set of acceptance records is called *acceptance set*. If the result of the evaluation produce the acceptance record (0, POLICY, R), the given assertions have proven that the request r complies with the policy.

In the example above, the acceptance records get from the credentials are (1, A, R) which means that "Alice approves the loan", and (1, A, R_B) which means that "Alice approves the loan if Bob approves it", and (2, B, R) and (2, B, R_A) similarly. In another paper clarifying how PolicyMaker works [9], a blackboard has been use as a metaphor for the evaluation of the assertions. In this example, the request is accepted, if the assertions are checked in the following order: first (f_1, A), (f_2, B), then again (f_1, A), (f_2, B), and finally (f_0, POLICY). First in checking the first assertion, a acceptance record (1, A, R_B) is attached to the blackboard. Then the second assertion is checked and acceptance record (2, B, R_A) is added to the blackboard. Still, no proof has provided for the question. The assertions are checked again, and acceptance records (1, A, R) and (2, B, R) can be attached to the blackboard because there is already information, e.g. that "Alice will approve the loan if Bob do so". Finally, the acceptance record (0, POLICY, R) can be added because both Alice and Bob has approved the loan, and the proof that the request comply with the policy has been shown.

Compliance Checking Algorithm

Now we know the notations for the compliance checker and I can present a compliance checking algorithm of Policymaker given in [12]. In figure 4.4

```
CCA_1 (r, {(f_0, POLICY), (f_1, s_1), ..., (f_{n-1}, s_{n-1})}, c, m, s) :
  {
    S ← {(Λ, Λ, R)}
    I ← {}
    For j ← 1 to mn
    {
      For i ← n − 1 to 0
      {
        If (f_i, s_i) ∉ I, Then S' ← (f_i, s_i)(S)
        If IllFormed((f_i, s_i)), Then I ← I ∪ {(f_i, s_i)},
          Else S ← S ∪ S'
      }
    }
    If (0, POLICY, R) ∈ S, Then Output(Accept)
      Else Output(Reject)
  }
```

Figure 4.4: Pseudocode for Compliance Checking algorithm [12]

is the algorithm that can solve if a given set of assertions proves that the request complies with the policy usually in polynomial time.

Let us return to the limitations for the compliance checker given in the beginning of this section. The algorithm in figure 4.4 is locally bounded and monotonic. The integer c in the input of the algorithm is the time limit for assertion evaluation, the integer m is the maximal number of action strings, and the integer s is the maximum size of an acceptance record. The monotonicity means that if an assertion is approved, also the subset of it will be approved. In addition to local boundaries and monotonicity, the algorithm is also authentic. This "authenticity" means that an assertion produces only correctly formed acceptance records and its issuer does not pretend to be someone else. The algorithm checks that every assertion obeys these limitations in the subroutine called IllFormed. All ill-formed assertions are ignored.

4.2 KeyNote Trust-Management System Version 2

Matt Blaze, Joan Feigenbaum and John Ioannidis from AT&T Labs Research and Angelos Keromytis from University of Pennsylvia are developed

the KeyNote Trust Management System. Second version of this system is published as RFC 2704 which is the main source of this section [8]. The design principles of KeyNote are that the system is simple, expressive and extensible [10].

The basis of KeyNote is similar to PolicyMaker: A principal wants to do an action in an application. The application has to check if the action fulfills its policy statements. It sends to KeyNote compliance checker a query that consists of application's policies, credentials of the principal and the action that the principal want to perform. The compliance checker of KeyNote advises the application what it should do, i.e. to reject or to accept the queried action.

Application must give its policies clearly and correctly, and it is responsible of fulfilling the decisions, KeyNote only gives advise based on the information that the application has given to Keynote. If credentials, policies, and the requested action are written correctly, the KeyNote will not approve an illegal action. Collecting needed credentials and policies is left for the application.

4.2.1 Components of KeyNote Trust-Management System

Similarly than in PolicyMaker, Keynote Trust-Management system has five components [8]:

- **Actions** and a language for describing these actions. An action has security consequences that the system wants to control.
- **Principals** can be allowed to perform an action. The trust management system has to have a mechanism to identify principals.
- **Policies** and a language for defining them. Policies control actions that principals are authorized to perform.
- **Credentials** and language for determining the credentials. A principal can delegate authorization to an another principal with a credential.
- **A compliance checker** determines if a principal is allowed to perform an action according the policy of an application and given credentials.

KeyNote is designed for Internet based applications [8]. Each application can have its own policy which is easy to change because the policy is passed

together with query to an external agent, KeyNote. When an application wants to check if a principal has a right to perform an action, it gives the requested action, policies, and credentials to a separate compliance checker. The compliance checker then informs the application if the principal has the right to perform the action or not. Next in this subsection, parts of the Keynote are presented.

Assertions

Like in PolicyMaker, KeyNote's *policies* and *credentials* are written with the same language and they are called assertions. An *assertion* describes the authorization given to a principal by another principal, and in what kind of situation the authorization is valid. The KeyNote assertions are designed to be human-readable but also computer-friendly.

An application has its own policy, denoted in policy assertions which are introduced later in this section. The application does not need to keep up an access control list because it can use credentials granted and signed by external authorities.

Assertions must be monotonic, i.e. assertions in a query given to be evaluated can only give more rights to a principal. This way, the principal cannot cheat the compliance checker by not giving all his assertions to it.

Actions

An application uses KeyNote to decide if someone is allowed to perform an *action*. An action is a collection of name-value pairs, an *action attribute set*. An application creates the action attribute set when it sends a request to KeyNote compliance checker. The compliance checker does not understand the semantics of the actions. Application developers and security administrators who writes the assertions need to agree on common language for the actions. The semantic of action defined in KeyNote specification is presented later in subsection 4.2.2.

Principals

Principals are the main actors in KeyNote. A principal can request actions and issue assertions. KeyNote's principal is presented by a string that can be a public key. Thus, the principal can sign assertions that gives the authorization of this principal to another principal.

KeyNote has a special principal which is identified with reserved word "POLICY". Like other principals, the POLICY principal can authorize other principal by granting *policy assertions*. The POLICY assertions are not signed, and they must be delivered through a trusted channel to the KeyNote. The POLICY is a root for trust, i.e. if another principal signs an assertion, this assertion is trusted only if it can be connected to an assertion that has POLICY as its authorizer.

Compliance Checker

The compliance checker informs an application how the asked action should be handled based on the given information. The answer can be "reject" or "approve" it the simplest case, but the compliance checker can also choose from given set of values. If the more complex version of the compliance checker is used, the application needs to arrange the possible answers to an order from the "weakest" right to the "strongest" right and send this information with the query to the compliance checker.

4.2.2 KeyNote Syntax

Action Attributes

Each action attribute has a name and a value. The interpretation of actions is decided by application, not by KeyNote. Policies and credentials have to have the same interpretation. By default, action attribute names and values are strings that can begins with any letter followed by letters and/or digits. The maximum length of an action attribute name is 2048 characters. It depends on the KeyNote implementation how the length of attribute value is expressed.

The names beginning with character `_` are reserved for inter KeyNote use. `_MIN_TRUST` means the lowest compliance value and `_MAX_TRUST` means the highest compliance value in a query. An ordered set of compliance values is given in attribute which name is `_VALUES` and list of directly authorized names is in `_ACTION_AUTHORIZERS` attribute. An application cannot use these as a part of a query.

Assertions

In this subsection, I first introduce the structure of an assertion in details and then give an example.

Assertions have their own structure which have only one mandatory and six optional fields. In the following structure definition, each nonterminal symbol is marked inside $< >$ marks and question marks represents optional fields. Each field begins with identifying label followed by : mark and a value of the field. Actually, each field begins a new line. An assertion has the following structure:

```
Assertion  :  <VersionField>?  <AuthField> <LicenseesField>?
              <LocalConstantsField>?  <ConditionsField>?
              <CommentField>?  <SignatureField>?  ;
```

An assertion always begins with the version field if it is present, and ends with signature if the assertion is signed. Other fields can be in any order. The `VersionField` consists of identifier "KeyNote-Version:" and the current version number that is two as either string or integer.

The `AuthField` is the only obligatory field in an assertion. It identifies a principal who has granted this assertion. The identifier of this field is "Authorizer:", and it is followed by principal identifier or a attribute id. Principals can be identified by a label which is opaque to KeyNote, i.e. it can be any ASCII string that does not contain : characters. Identifier can also be an encoded cryptographic key. A key identifier begins with the name of the used algorithm (e.g. RSA, DSA, etc.), then after ":" character is the encoded (e.g. BASE64 or HEX) key itself.

The assertion gives authorization to a principal identified in the `Licensees-Field`. One assertion can give authorization to several principals with conditions such as both of these principals (e.g. ("RSA:alice_key" && "RSA:bob_key")), some of these principals (e.g. ("RSA:alice_key" || "RSA:bob_key")), or k of the following set of principals (e.g. 2-of("RSA:alice_key" ,"RSA:bob_key", "RSA:carol_key").

The `LocalConstantsField` gives shorter names, for example, for keys used in this assertion, which makes the assertion more readable for humans. Especially, shorter names make Licensees field more readable. The Local-Constants field consists of identifier "Local-Constants:" and a list of "attributeID = string" pairs. The attributeID is the used alias, and it is a string as defined in the previous subsection about action attributes. After the "=" character is the replaced string inside quoting marks. If this string is a public

key, the string contains the name of the used algorithm in addition to the actual key.

The authorizer can set conditions under which the authorization is given to the licensees. The conditions are in the `ConditionsField` which begins with "Conditions:" following by a list of "test -> value" conditions. Tests can be comparisons of numbers or strings, and optional values are strings.

The `CommentField` denotes the purpose of the assertion. It consists of string "Comment:" followed by ASCII text. Other way to insert comments to assertions is insert them anywhere in the assertion. Then the comment begins with #-character, and the comment is written in ASCII. Comments are ignored by KeyNote, but they are included in the assertion signature calculation.

The `SignatureField` identifies the algorithm used to sign the assertion. This fields begins with "Signature:", then follows the algorithm identifier, ":" character, and finally the signature. The signature is calculated from the whole assertion including comments but excluding the SignatureField.

For example [8]:

```
KeyNote-Version:  2
Comment:  A simple email certificate for user Alice
Local-Constants:  CA_key ="RSA:abcdef...1234512345"
  Alice_key = "DSA:abcdabcd...abcdabcd"
Authorizer:  CA_key
Licensees:  Alice_key
Conditions:  ((app_domain == "email") #valid for email only
  && (address == "alice@example.com"));
Signature:  "RSA-SHA1:  f00f2244"
```

4.2.3 KeyNote Queries

When an application wants to know if an action is allowed by its policy and given credentials to a principal, it sends a query to KeyNote. The query has four parameters [8]:

- The identifier of the principal who wants to do the action
- The action attribute set that describes the action
- The set of possible compliance values ordered by the application from _MIN_TRUST to _MAX_TRUST

- The policy and credentials that need to be noticed when the query is evaluated.

Policy assertions are unsigned, and they are trusted because the policy is part of the trusted application and they are provided through trusted channel to KeyNote. Other assertions, credentials, are signed by the authorizer (a principal). Either the application or KeyNote can verify the signatures of the credentials because principals are usually public keys. If KeyNote knows the cryptographic algorithm of the public key, it can verify the signature of the assertion. If the algorithm is unknown, the signature cannot be verified by KeyNote and the the assertion is treated like a trusted policy assertion, i.e. the application must take care of the verification. Because credentials are signed, the application can send them to KeyNote over untrusted channel.

The answer can be a boolean value, like reject or approve, but some applications use more options in the evaluation of authorization, for example no access, limited access, and full access. The possible answer values are given to KeyNote as an ordered set from the lowest value to the highest value. A principal will get the highest trust value _MAX_TRUST, if it is the authorizer of the action in the query, or if there is no conditions field at all. If the conditions field is empty but present, the principal will get the minimal trust value, _MIN_TRUST. Otherwise, the trust value is calculated based on the conditions and the licensees as presented next in this subsection.

The `ConditionsField` gives a list of "test -> (optional) value" elements. The test either succeeds or fails. The value is a string, one from the compliance values given in the query. If the value is missing, it is considered to be the _MAX_TRUST. Each test in the condition field gives its own value. The condition compliance value will be the highest of these possible values. For example [8]:

```
Conditions:
  user_id < 1000 -> "user_access";
  user_id < 10000 -> "guest_access";
  user_name == "root" -> "full_access";
```

In this example, if the query has given value "1073" for the "user_id" attribute and the user_name is "root", the possible values are "user_access" and "full_access". The condition compliance value is the highest from these two values. If the compliance values of the query are {"no_access", "guest_access", "user_access", "full_access"}, these conditions has given "full_access" in this example.

Each query has at least one requesting principal. If several principals requests the action, their identifiers are included in the query, and the query is first evaluated separately for all of the principals. In the end, the result is combined together according to the rules given in the `LicenseesField`, and the result is returned to the requesting application. For example [8]:

```
Licensees:  (“alice” && “bob”) || ”carol”
```

First, the conditions are checked separately for each principal, for alice, bob, and carol. If alice is allowed to perform the action and others are not, the licensees compliance value will be the lowest value in the compliance value set, e.g. the principals are not allowed to perform action. For evaluating the licensees compliance value, following rules are used:

- An expression given in parenthesis, (...), will get the value of the subexpression inside parenthesis
- An expression “A && B” will get the lowest trust value of the subexpressions
- An expression “A || B” will get the highest trust value of the subexpressions
- In evaluating a “K-of-(a list)” expression, the compliance values got from the evaluation of each list members are ordered to a list from the minimal trust value to maximal trust value. If several principals get the same value, the value will be in the list several times. The licensees compliance value is the Kth value from the result list.

4.2.4 Differences between PolicyMaker and KeyNote

KeyNote is based on PolicyMaker, and their approach to the trust management problem is similar. Matt Blaze and Joan Feigenbaum have participated to the development of both these systems. Even though the basis is same, there are differences. These differences are presented in table 4.1. The main differences are in the used languages, the roles of the application and trust management system, and what kinds of answers the trust management system returns.

	PolicyMaker	**KeyNote**
Return value	boolean or restrictions that could give the positive answer	boolean (authorized or not) or value from a list
Signature verification	application	KeyNote itself
Assertion syntax	AWKWARD programs, regular expressions	RFC-822 (email) -style human readable syntax, c-like regular expressions

Table 4.1: Central differences between PolicyMaker and KeyNote [10]

4.3 Simple Distributed Security Infrastructure (SDSI)

Simple Distributed Security Infrastructure (SDSI) is a simple public key infrastructure which defines authorization certificates. First version of SDSI was defined in 1996 by Ron Rivest and Butler Lampson at Massachusetts Institute of Technology [77]. Later, SDSI is joined together with Simple Public Key Infrastructure (SPKI), and the second version of specification introduces this merged version of SDSI/SPKI [38]. In this thesis, I first present the old version of SDSI because it introduces new ideas for decentralized authorization. Thus, this section is based on the SDSI version 1.0. The next section describes the current merged version of SDSI/SPKI under name of SPKI because SPKI is standardized by IETF.

4.3.1 SDSI Terminology

SDSI is a "key-centric" system [77] where everything is based on public key cryptography. A *principal* can be a person, enterprise, machine etc. and it is represented by a public key. Each principal can act as a certification authority and give digitally signed statements about another principals i.e. there is *no global hierarchy* of certification authorities. SDSI uses *reconfirmation* instead of certificate revocation lists. This means that validity of a certificate is checked from the principal who has granted it.

What makes the idea behind SDSI unique, is a concept of *local name spaces*. Each principal decides by herself what to call other principals. For example, Alice can be different person in one principals name space than in another name space, and it can even be, for example, a program in someone's name space. There is no global name space but there are few *distinguished root*

principals whose name space is the same to all principals. Usually, global names are not used. But how we can distinguish all different Alices? In SDSI, the local name spaces are *linked* together. We can say than we mean that Alice who is defined in Bob's name space instead of Alice that Carol knows.

SDSI certificates can be *identity certificates* that binds together a local name and a public key. When creating identity certificates, the principal must carefully and manually check that the identity is correct. In addition to certifying identities, SDSI certificates can denote *authorization* or *group membership*. As can be remembered, an authorization certificate binds a public key to an action which is authorized for the owner of the corresponding secret key. In SDSI, an authorization can be *delegated* to another principal. The delegation can be done with *delegation certificates* or *groups*. The delegation certificates are primarily used for authorizing a group of servers to sign certificates on behalf of a principal. The server gives this certificate as a proof of authorization to a querier.

Any principal can define a group that consists of principals or other groups. The group has a local name in the name space of the principal that has defined it i.e. the owner of the group. Groups are one way in SDSI to denote *roles*. In the paper [77], Ronald Rivest and Butler Lampson give an example where a hospital administrator creates a group which is named "head nurse". The group consists only the name of that principal who is acting as the head nurse at that working shift. The problem in this kinds of roles is that, for example, when the head nurse needs to sign a paper, she signs it as herself and not in the role because groups do not have public keys. Other way to denote roles in SDSI is to create for each role an own principal which is, as mentioned above, represented by a public key.

4.3.2 SDSI Data Structures

According to its name, SDSI uses simple data structures. Even complex data can be represented efficiently with S-Expressions defined by Ron Rivest [75] in human readable ASCII form. An S-Expression is an octet string or a list of simpler S-Expressions. A sequence of eight bit long octet is an octet string that can be represented as [75]

- a **token** that is a string of printable characters such as abc
- a **quoted string** which contains blanks, e.g. "Bob Smith"

- a **hexadecimal string** which begins with # character and contains only hexadecimal digits, e.g. #123abc
- a **verbatim octet string** that has two parts separated with a colon, the length of the string and special characters that are not available in other form of octet strings such as ^ , ~ , < etc. For example, #03:;?%
- a **base-64 encoded** octet string or verbatim octet string, for example |YMJj| and {MzphYmM=} which both are encoded string abc.

Because SDSI data structures are sometimes read by people, SDSI provides presentation hints. These can be used by the presentation program to decide how the data is shown to a user. SDSI documentation defines that simple MIME Content-Types can be used as presentation hints [77]. A presentation hint is an octet string surrounded by square brackets, for example [application/postscript] is a presentation hint for a MIME Content-Type.

Usually SDSI objects are lists. The first element is the type of the object and then the list consists of attribute/value pairs. The object type is a token ending with a colon. Attributes may have one value or list of values. Attributes can be in any order but the last attribute in the object is usually signature. Signature is also the only attribute that can be in an object several times, other attributes either are or not in the object. For example, here is a SDSI object that is a SDSI principal:

```
( Principal:
  ( Public-key:
    ( Algorithm:  RSA-with-SHA-1 )
    ( N: =Gt802Tbz9HKm067= )
    ( E: #11 ) )
  ( Global-Name:  ( ref:  DNS!!  fi hut Sanna.Liimatainen ) )
  ( Server-At:  "http://one.hut.fi/cgi-bin/sdsi-server/" )
  ( Principal-At:  "http://two.hut.fi/cgi-bin/sdsi-server/" ) )
```

This example contains an exception for the above mentioned structure: There is one object, the public key, directly inside another object. This saves little space because `Public-Key` string is mentioned only once. The example introduces a principal and it's public key. Other key types are `Private-Key` which is the corresponding to a public key and used for signing and decrypting, and `Shared-Secret-Key` which can be used in symmetric cryptography to encrypt and decrypt messages and as an one time password. In this principal, `N:` is a base-64 encoded RSA public key modulus and `E:` the used RSA exponent as a hexadecimal number.

In addition to a public key, the principal object in above example contains an optional global name (there can be several global names in a principal) and optional Internet addresses. Global names are in `Global-Name` attribute, and they start with a distinguished root. SDSI has reserved the following names for the distinguished roots: `VeriSign!!`, `IAPR!!`, `USPS!!`, and `DNS!!`. As can be seen from the list above, they are separated from other principals with double exclamation marks. The `Server-At` attribute gives an Internet address for a server that can give additional certificates for the principal and the `Principal-At` attribute gives an Internet address where one can send questions to the principal if the principals servers do not answer.

A principal object should consist only of the minimal information and all external information should be stored to principals "auto-certificate". There are two reasons. The principal object is included to every object signed by the principal which will take more space. Other reasons is that if changing information is in an auto-certificate, the principal object does not need to change. The auto-certificate is signed by the principal itself and it gives additional information about the principal mainly for people, for example a picture. It is not trustworthy because there is no third party to confirm the data stored in the auto-certificate. In SDSI, auto-certificates are denoted with `Auto-Cert:`.

An object can be encrypted. There are two ways to handle encrypted objects. An object whose type is `Encrypted-Macro:` will be decrypted immediately when it has been received and the clear text version of the object will be stored. Objects of type `Encrypted:` are stored in encrypted form and only decrypted when they are needed.

When speaking of public key infrastructure, signing of objects is one key function. SDSI objects can contain the signature of the object as the last attribute/value pair. Another way is to give a separate signed-object which contains the signed object itself or a hash of it. A signed-object consists of an object hash specifying the signed object, date (and time) when the signature is created, and the signature that is the result of a signature algorithm whose input was the signed-object without the signature attribute. An example is below [77].

```
( Signed:
  ( Object-Hash:  (SHA-1 =7YhdOmNcGFE071QTzXsap+q/uhb= ) )
  ( Date:  1996-02-14T1:46:05.046-0500 )
  ( Signature:  #3421197655f0021cdd8abc21866b )
  ( Expiration-Date:  2003-01-01-00:00:00.000-0500 )
  ( Signer:  ( Principal:  ...  )  )
  ( Re-confirm:  P1M ( Principal:  ...  )  )
  ( Credentials:  ...  )
  ( Signing-For:  ( Principal:  ...  )  )  )
```

As we can see from the example, the signed-object may also contain other information such as an `Expiration-Date` after which the signature is no longer valid. A `signer` attribute is necessary if the signer may later be unclear or signature needs to be reconfirmed periodically. The `Re-confirm` attribute give a time period (marked with `P`) in years (`Y`), months (`M`) like in this example, days (`D`), hours (`H`), minutes (`M`), and seconds (`S`). The months and minutes are separated because if the period is given in hours, minutes and seconds, before them is a letter `T`. After the time period, there is a principal to whom the reconfirmation is asked and which usually is a server. A `Signing-For` attribute tells that signature is done on behalf some principal. The `Credentials:` attribute gives credentials that show that the signer has authorization to sign on behalf of that other principal.

Each principal can refer to names in its local name space in two ways: either with a token such as Alice or Alice@example.com or with S-Expression with form (Local-Name: n). Local names can have some value which is bound to it with a certificate issued by a principal. For example [77],

```
( Cert:
  ( Local-Name:  foo )
  ( Value:  ( Principal:  ...  )  )
  ( Description:
     [text/richtext]
     "Bob Smith has worked at <bold>Bar Inc</bold> since 1995."
  ( Signed:  ...  )  )
```

is a certificate that binds local name foo to a principal that probably presents Bob Smith because in the `Description:` mentioned for human reader give a statement about him. The purpose of the description is to give extra information in an identity certificate. A `Value:` field in a certificate have to be `Principal:`, `Group:`, `Quote:` or `ref:` that is used to refer name space defined by someone else, or it can be a `Encrypted:` object from the previous list or an `Assert-Hash:` whose first field is from the list. The `Assert-Hash:`

is an object that reliably defines a name defined in a `ref:` object. It contains a the name and a hash value of it.

This subsection has introduced the syntax of SDSI objects. The next subsection tells how these objects can be used in authorization in different kinds of situations.

4.3.3 Communication with SDSI Servers

In addition to defining certificates, their structure and use, the SDSI specification defines communication protocols for getting information from SDSI servers [77]. Actually, SDSI is a framework and it can be used to define more message exchanges than presented in its specification. Two main tasks of SDSI servers are to act as a certificate repository where needed certificates can be looked up, and because SDSI does not use certificate revocation lists, the servers are needed to reassure certificates. This subsection introduces messages and their exchanges in both cases.

SDSI specifies a protocol called `Get` for searching certificates from a server database. The query always contains a principal (in a `To:` field of the query) whose signature needs to be in all certificates that are sent as an answer to the query. In addition to the signer, a query may contain a "template" that specifies object types, attributes and values for the search. Rivest and Lampson give two examples of queries [77]:

```
( Get.Query:
  ( To:  ( Principal:  ...)  )
  ( Template:  ( Cert:  ( Local-Name:  jim ) ) )
  ( Signed:  ...  )

( Get.Query:
  ( To:  ( Principal:  ...  )  )
  ( Template:  ( Cert:  ( Value:  (Principal:  ...  )  )  )  )
  ( Signed:  ...  )
```

In the first example, a certificate for jim is searched, and in the second example, the searched certificate contains a certain value. In addition to `To:` and `Template:`, the query can contain mandatory or optional wishes about the reply: that it should be encrypted or/and signed. Whether the reply is signed or not, each certificate contains a signature.

The reply for the query contains a hash from the query, discovered certificates and a signature if asked. The signing of the whole reply message protects it from tampering, for example, deleting of a certificate from a list of found

certificates. An error message also contains a hash of the original query and give some description of what error has occurred. The error message might also be signed by the server.

SDSI defines other type of query for membership certificates. A principal can ask if she is a member of a group with a `Membership.Query''`. The reply does not return a list of all the members of the group, it just say if the principal(s) are the members of a group or not, or if the query failed, a hint for the reason may be given. The reply is signed by the server, and thus they can be used as credentials denoting some authorization.

Certificates contain information about how their validation should be done. Some certificates requires periodic reconfirmations specified by the signer. The original signer of the certificate can reconfirm the certificate, but also a reconfirmation principal (a server) who has got the authority from the original signer can make the reconfirmation. A reconfirmation query contains a field telling the supposed signer of reconfirmation and a signed object that needs to be reconfirmed. Similarly to the above, the reply has a field for a hash of the query. The signed object has been signed again and the new signature is attached to end of the object. The new signature may have an expiration date but it does not have reconfirmation property.

4.4 Simple Public Key Infrastructure (SPKI)

The purpose of Simple Public Key Infrastructure (SPKI) Working Group in IETF was to develop simple and extensible method for trust management. The main idea of SPKI certificates is to give an authorization to a keyholder. The keyholder is an entity who has a certain public key and the corresponding private key. SPKI does not specify where these certificates are stored and how they are distributed.

The development of SPKI started in the IETF in year 1996. A part of the work has been published as experimental RFCs [27, 29]. An experimental RFC means that the work is published as general information and does not specify any Internet Standard [18]. The rest of SPKI are published as Internet Drafts but they are already expired [28, 31]. In this section, we familiarize ourselves with SPKI certificates and their use and management.

4.4.1 SPKI Certificates

There are three kinds of certificates: identity, attribute and authorization certificates. As mentioned earlier, an identity certificate binds a name to a key, an attribute certificate is used to map authorization to a name, and an authorization certificates ties authorization directly to a key. SPKI certificates are designed to be used as authorization certificates. Thus, SPKI certificates may be used to give an authorization to an anonymous public key without knowing who the keyholder is. The authorized user is the keyholder who knows the corresponding secret key. The keyholder can even be a program or a server, not a person at all. Certificates of Simple Public Key Infrastructure are designed to be as simple as possible. Each certificate contains information just as much is needed to give the authority. One purpose of simplicity is that the certificates do not reveal the identity of the keyholder even when several certificates are presented together.

The content of a SPKI certificate is described as a 5-tuple. The actual certificate also contains other fields, for example a version number, and it is digitally signed by the issuer. RFC 2693 defines the elements of a 5-tuple as follows [29]. All the elements are described below in more details.

- **Issuer** is the initiator of an authorization. It is presented by a public key or a hash of a public key.
- **Subject** is also a public key or a hash of a public key but it may also be a name or a hash of an object. It represents the entity to whom the authorization is given.
- **Delegation** is a boolean. If delegation is true, the subject can further delegate the authorization to someone else.
- **Authorization** is the given right expressed by a S-Expression (described in details above in subsection 4.3.2).
- **Validity** usually tells "not-before" and "not-after" dates and times. SPKI also provides the possibility to use three kinds of online validations.

Issuer and Subject

Applications which offer some service or resource usually do not need to know the identity of a user. It is enough to know if the user is authorized to do

the requested action. The SPKI certificate usually maps an authorization directly to a public key, the mapping is not done through a name like in X.509. The issuer field always contains a key or a hash of a key (or the reserved word "self") but the subject field may also be a name. When an authorization is given to an entity, for example a program which can be download from a WWW-page, the subject field contains a hash of the entity object.

In section 2.2, problems of globally unique names are discussed. In SDSI, Rivest and Lampson introduced how local names can be used globally [77] (SDSI and local name spaces is discussed in more detail in section 4.3). SPKI uses the name structure of SDSI and later these two PKIs are merged. Computer environment usually needs globally unique identifiers even when local names are used. Public keys of public key cryptosystem can be used as such because they must be unique and they can be published [29]. A hash value calculated by collision-free hash function can also be used as an identifier.

Delegation

In SPKI, boolean control of delegation is used [29]. The owner of a resource can give to the subject the right to further delegate the authorization. If the owner of a resource gives the delegation right, she cannot limit the delegation, i.e. the subject can decide if she want to give the authorization to someone else. In the long run, the resource decides based on given proofs (certificates) if the requested action is allowed. With boolean control of delegation, the absence of delegation right denotes that delegation is not allowed.

When Ellison *et al.* were developing SPKI, they also considered two other kind of delegation control: integer control and no control. Integer control which can be used to restrict the depth of delegation is more complex. The owner of a resource must somehow predict what depth of delegation is needed and this is difficult in an environment where a user can create new keys. If the delegation is not allowed at all (no control case), the subject can feel temptation to loan her private key to someone. This violates the principle of public key cryptography where private key must be kept secret.

Authorization

Formation and interpretation of SPKI certificates must not require large computing capabilities because it should be possible to handle these certificates

in a smart card or other similarly small device. Certificates are defined with S-Expressions which are simpler ways of representing data than, for example, ASN.1. Rivest has developed an application independent S-Expression construction and utilization [75].

Validity

Usually validity is a time period that is defined by not-before and not-after dates and times. This is not always convenient. For example, what can be done if the keyholder losses her secret key? SPKI offers three online validation methods: certificate revocation lists (CRL), revalidation and one-time certificates.

When a CRL is used, a SPKI certificate has information where the CRL can be found and who (which key) must have signed it. There can be several locations for the CRLs. The SPKI specification also gives definitions to the CRLs: The CRL list must have a validity period and there must be only one valid CRL at a time, i.e. when a new CRL is published, its validity period must not to be overlapping with the old validity period. When CRL defines certificates that are not valid, the revalidation tells if a certificate is still valid. Because time interval in certificate validity field or in CRLs cannot be zero, other method is needed to tell if a certificate can be used only once. One-time validation is based on signed nonce which prevents using a certificate several times.

4.4.2 Syntax of SPKI Certificates

SPKI Authorization Certificates

Syntax of SPKI authorization and name certificates is defined in an expired Internet draft in 1999 [28]. The draft is about the merged SPKI/SDSI version 2.0, and objects are defined as S-Expressions familiar from above examples in SDSI section of this thesis. First, the authorization certificates have the following form [28]:

```
cert  :  (  cert <version>?  <cert-display>?
            <issuer> <issuer-loc>?  <subject> <subject-loc>?
            <deleg>?  <tab> <valid>?  <comment>?  );
```

Question mark means that the field is optional. The nonterminal symbols are given inside $<$ and $>$ and these are replaced with the actual text in the certificate, usually given in parenthesis. For example, the first optional field

is a `<version>`. If the field is not present, the version is assumed to be `( version 0 )` i.e. the version field contains the word "version" and an integer in parenthesis.

A `cert-display` field gives hint to a user interface of an application how to present the certificate. A `comment` field is for human readers. Both cert-display and comment fields are ignored when certificate code is processed.

First obligatory field in the certificate is a `issuer` field. It contains the word "issuer" and a public key or a hash of a public key of a principal. A `subject` field is similar, but there are more alternatives. In most cases, the subject is also a public key or a hash of one. The subject can be also a relative or fully-qualified name of a principal or a group, or a hash of an object that is not a principal but that is, for example, a program. For certificates that are for human readers, the subject is marked with keyword `keyholder`. It means the actual owner of the public key in flesh and blood/iron and silicon and usually the certificates refer to the real world giving, for example a physical location of a server or birthday of the keyholder. Last possibility for the subject field content is a complex k-of-n construction, for example "three keys from the following list".

An optional `issuer-loc` field contains a keyword "issuer-info" and gives an URL of a server or other entity which has additional certificates for the issuer. This location may be necessary for getting complete certificate chains: a server may have a database from where one can fetch the certificates that tell how the issuer has got the right to give the authorization to the subject. Similarly, an optional `subject-loc` field gives location of additional information. For example, if the issuer or the subject is a hash of a public key, in the additional field can be a location of a server that has the whole public key.

The subject of the certificate can delegate the right or part of the right given in the certificate to someone else, if the `deleg` field contains a word `propagate`, otherwise the delegation right is not given to the subject.

The `valid` field gives the dates when the certificate is valid. If both are missing, the presumption is that the certificate is valid forever. If one of them is missing, the presumption is forever but the other date restricts the validity. In addition or instead of validity dates, the valid field can give online validation service that is used. The validation can be based on certificate revocation list or valid certificate list in a given server or an one-time signed response to a certain challenge. The valid field can also be a kind of a request for a new short time certificate from a given server.

The last field of an authorization certificate is an obligatory `tag` field. It defines a permission given by the issuer of the certificate to the subject. The permission is defined with S-Expressions. If the tag is `(tag (*))`, it means "all permissions". The permissions can only be reduced from that, i.e. if something is added to the permission, it must restrict the permission. For example, Alice gives to Bob a right to access all files in her WWW home directory and a right to delegate the certificate. Bob can now write a certificate that restricts the access right to be only index.html file in Alice's WWW directory and gives the certificate to Carol.

SPKI Name Certificates

Other form of specified certificates is SPKI name certificates [28]. In authorization system based on public keys, the names are only helping humans to handle things. Anyone can issue name certificates that binds a public key to a name that might have significance only to the issuer of the certificate. This is possible, because as we can remember, SPKI/SDSI uses local name spaces. A name certificate has a following form:

```
cert  :  (  cert <version>?  <cert-display>?  <issuer-name>
            <subject> <valid>?  <comment>?  );
```

`Version`, `cert-display`, `valid`, and `comment` fields are same than in an authorization certificate. A `tag` and a `deleg` fields are not needed because a name certificate just gives a name for a public key. The `issuer-name` field will be evaluated to `( issuer (name <principal> <name>))` where the principal is the issuer of the certificate and the name is the name she has given to the subject. The name given to the subject can be from a local name from any name space or it can be a fully-qualified name. A name itself is a byte string. The `subject` can be virtually anything that will be given a name by this certificate. If several name certificates give the same name to different subjects, the name is given to a group.

Next I tell how SPKI certificates are used and give examples about their use.

4.4.3 Use of SPKI Certificates

The main purpose of SPKI is that an owner of a resource can give authorization to use the resource to keyholders by issuing SPKI authorization certificates. The resource owner can also give a right to the keyholder to further delegate the authority. SPKI certificates should be as simple as possible and usable in different kinds of systems, for example in smart cards. Still, the

SPKI certificates should be suitable for several kinds of applications, such as transactions with banks and denoting ownership of immaterial property [27].

Example

When a positive access control decision is based on SPKI certificates, the certificates must form a chain from the owner of the resource to the user of the resource with correct authorization fiels. Let us consider the above example of Alice's WWW homepages in more details. First, Alice has given to Bob a right to access all files in her WWW pages and a right to further delegate this right. She has issued a SPKI certificate to Bob:

```
(cert
  (version 0)
  (issuer
    (hash sha1 #1a6f6d62 1abd4476 f16d0800 fe4c32d0 6ff62e93#))
  (subject
    (hash sha1 #1a2b3c4d 1a2b3c4d 1a2b3c4d 1a2b3c4d 1a2b3c4d#))
  (subject-info http://www.example.com/bob/bobkey.html)
  (propagate)
  (tag (http (* prefix http://www.alice.example.com/)))
  (not-before "2000-01-01_00:00:00")
  (not-after "2002-12-31_23:59:59"))
```

In the above certificate, the issuer is denoted by a hash of Alice's public key and the subject is a hash of Bob's public key. This certificate also indicates where Bob's public key can be found. The certificate gives a right to access all the files in Alice's WWW home directory which address is given in the tag-field. Alice has given that right and its delegation rights to Bob for three years. Now, Bob can create another SPKI certificate for Carol:

```
(cert
  (version 0)
  (issuer
    (hash sha1 #1a2b3c4d 1a2b3c4d 1a2b3c4d 1a2b3c4d 1a2b3c4d#))
  (subject
    (hash sha1 #12345678 abcdefab 12345678 abcdefab 12345678#))
  (tag (http (* prefix http://www.alice.example.com/secret/)))
  (not-before "2002-05-01_00:00:00")
  (not-after "2002-08-31_23:59:59"))
```

This certificate give more restricted access rights and for shorter time period than the original certificate, and Carol cannot further delegate the rights she

has got from Bob.

When Carol wants to access a file in Alice's WWW pages, she gives two certificates as a proof of the authorization: the one given to Bob and the one given by Bob. These two certificates form a chain from Alice, who has given the original right to Bob, who has restricted the rights he has given to Carol, who wants to perform an action that she has authorization to perform. But both certificates have different validation times and different authorization content.

The certificate chain is reduced in validation to one *certificate result certificate* that presents the rights given by a set of certificates. The authorization and validity of this new certificate are intersections from the corresponding fields of the original certificates. The intersection of the authorization fields is done by checking them element for element. Each element restricts the authorization. If the authorization field of one certificate has a longer list of elements and the shorter list is part of the longer one, the authorization field of the certificate result certificate will contain this longer list of authorization elements. If the authorization elements are different in original certificates, the intersection fails and no authorization has been given based on the certificate chain.

The validity time is the shortest time the certificates gives, i.e. the validity time starts from the latest time and ends in the earliest time given in the original certificates. But the time is not the only validity controller, certificates may also require online validity tests. The online test may give similar time period than the certificates themselves, and the validity time calculation is similar than above. Otherwise, the validation process will ask from all online servers about the validity of each certificate in the end of the validation process.

In the above example, the certificate result certificate is similar from its authorization and validity time than the certificate given by Bob because Bob has restricted the authorization and validity times from the certificate he has got from Alice.

Certificate Path Discovery

The basic assumption of SPKI is that the requester has all the certificates she needs in order to prove her authorization. The task of the verifier is to check if the set of certificates forms a proof that the authorization can be allowed to the requester. Dwaine Clarke *et al.* suggest following algorithm for finding certificate chain [22]:

1. **Remove useless certificates**, i.e. invalid certificates and certificates which do not authorize requested action.

2. **Calculate name reduction**, i.e. find all certificates that are reduced.

3. **Remove all names and name certificates** because they just give an identifier to a key.

4. **Remove useless authorization certificates** that cannot be part of the chain.

5. **Use depth-first search fo find a path** from a graph formed from certificates: nodes are keys and edges are authorizations.

6. **Reconstruct the certificate chain**

Key Management

"Key management is simplest when all cryptographic keys are fixed for all time" denote Menezes *et. al* [61], section 13.7. In practice, cryptographic keys must be changed time to time for several reasons. Finding an encryption key with cryptoanalysis become easier, if large amount of text is encrypted with the same key [79]. Furthermore, keys can be stolen or passphrase for using the key can be forgotten. Because public keys are used to represent entities in authorization certificates, the certificate becomes invalid if the key is invalid.

SPKI has no separate method for key management. It is handled with certificate validity [29]. If a key has compromised, the certificate giving an authorization to it can be revoked. Revoked certificates are published in a certificate revocation list. The validity field of the certificate can require the use of online validation, and give the location of the used CRL. Other way to enforce the change of keys is the validity time period of a certificate. When the certificate become obsolete, a new certificate for a new public key can be issued.

Some applications may use names in access control lists and name certificates to bind the names to keys. But names are problematic and there is no unique names as such. The names are used together with a public key. The lifetime of a key is usually shorter than the life time of names. Using of a master key can make lifetime of keys, and names, longer. The master key is used to delegate all authorizations to other key that have shorter lifetime, and this other key is used when needed.

4.4.4 Access Control List in SPKI

Often a service keeps a list of legitime users and their rights. SPKI supports also this traditional access control method. Because the access control list is internal to the service, an access control list entry can be any kind. It is used only by the service, and not given to others.

Although the implementation of access control is free for the developer, SPKI gives a sample implementation for access control entries. The entry can be same form as a SPKI certificate but the issuer field and the signature are unnecessary. The access control list is usually stored in the same machine than the resource which it manages, and the list is issued by a trusted administrator.

Authorization certificates and their delegation release the service from keeping up an access control list. The service does not need storage for the ACL, and it can be simpler. The access control management is distributed outside from the service. Delegation is simpler because it does not need to be checked before the request to use the service.

4.4.5 Certificate Revocation

The validity field of the certificate defines if the certificate revocation list is used for checking valitidy of the certificate. As mentioned above, the field gives the location of the certificate revocation list and the public key that must be used to sign it.

Eliminating CRLs

Ronald Rivest argues that certificate revocation lists should be eliminated [76]. He states that if certificate is issued recently and suplies all needed evidence of the authorization, and the expiration date has not yet been, certificates can be used as such without certificate revocation lists. New certificates are best evidence of authorization, and they are easy to provide.

The certificate revocation lists is an approach from the wrong direction. The acceptor of a certificate should be the entity which makes the decisions if a certificate is valid, not the issuer of the certificate [76]. Only the acceptor can decide, for example, how old certificates are suffieciently new.

The receiver, which verifies the certificate, can be simpler if all necessary information about authorization is stored in a certificate. Otherwise, the

verifier needs to search for additional information which causes more work for the verifier. For example, a popular service can get overloaded if it must search more evidence for intentionally forged certificate chains (denial-of-service attack).

One reason for the use of CRLs is that the keyholder loses her private key and some method for announcing about the loss is needed. Rivest suggests that CRL can be replaced by a "key compromise agent". These key compromise agents will form a network. If a key has become unvalid, the information is told to the nearest agent that then distributes the information to other agents. The needed information is that the key has *not* been compromized rather than that it has been compromized. The key compromise agent issues "certificate of healt" that denotes that none of the agents has not heard that the key is compromized. This certificate of healt is given to the verifier together with the other certificates.

In the last section of this chapter, I describe an architecture that uses Simple Public Key Infrastructure for trust and policy management, but adds also other needed components to be able provide authorization and delegation in distributed agent system.

4.5 Telecommunication Software Security Architecture

Telecommunication Software Security Architecture (TeSSA) is developed by Pekka Nikander and his research team in 1998 at Helsinki University of Technology [67]. Differently from the trust management systems presented earlier, TeSSA is an architecture for secure distributed systems and agent systems.

Agent system consists of computing nodes, insecure network, and software agents [67]. Computing nodes are capable to execute software agents. An agent is a piece of code and data. The purpose of the TeSSA architecture is to denote trust that exists in agent systems and make the execution of agents secure for both the agent and the node.

As presented in the figure 4.5, the TeSSA architecture has four parts: session security protocol, authentication or authorization protocol, certificate repository, and trust and policy management infrastructure. In the TeSSA research project, the architecture was implemented and the implementation is based on Internet Protocol Security (IPsec) [52] for providing session security and

Internet Security Association and Key Management Protocol (ISAKMP) [60] for authentication. The Domain Name System (DNS) [62] serves as a certificate repository for SPKI certificates that are used for trust and policy management. This section represents the TeSSA architecture and describes how the architecture can be used to secure distributed systems.

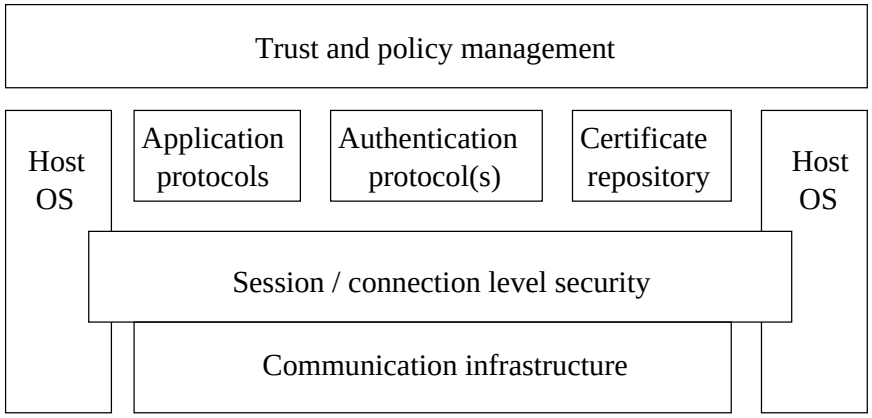


Figure 4.5: The conceptual building blocks of the TeSSA [67]

4.5.1 Building Blocks of the TeSSA Architecture

The basic building blocks for the TeSSA architecture to work in the TCP/IP environment were implemented in the TeSSA project [25] and in its predecessor, the IPsec project [26] at Helsinki University of Technology. This section represents how the architecture was implemented for the Internet environment starting from the lowest security level in the figure 4.5 and ending up to the highest level, to the trust and policy management.

Session Security

The implementation of the TeSSA architecture uses the *Internet Protocol Security (IPsec)* [52] to ensure session security. The IPsec is presented earlier in section 3.2.1. It gives two basic ways to provide security: AH for integrity and ESP for confidentiality. Each node, a host, in a network should have its own security policy that defines what kind of protection of connections is required.

The session level security is tightly connected with the access control of the operating system [67]. I describe below in this subsection how this connection is created in the TeSSA architecture.

Authentication Protocols

Communicating parties must first agree on how a connection is protected. IPsec offers two possibilities for connection establishment negotiations. A simple protocol called the *Internet Key Exchange (IKE)* [43] is one protocol for negotiation of security associations, but the TeSSA architecture does not use it because it does not offer enough choices [67]. The Tessa Architecture uses the *Internet Security Association and Key Management Protocol (ISAKMP)* [60] because it defines extendible framework for negotiation of security associations for different kinds of purposes. In ISAKMP negotiation, the initiator of the connection proposes methods to protect the connection by giving a set of security associations that represent its security policy. The responder chooses one security association that suites for his security policy.

The standard IPsec security policy is not enough for the TeSSA architecture [67]. It defines only how individual datagrams are protected and ensures that the datagrams end up to the correct application. The TeSSA architecture connects the local access control policy to the communication policy. TeSSA defines three roles for access control and security policy: how new connections can be created, how the connections must be protected, and what kind of request can be send over the connections [67]. The policy control of the first two is implemented with ISAKMP but the last restriction is not possible to determine with the ISAKMP protocol. The third part of the policy that binds the session level security to the access control is implemented SPKI certificates.

Before I introduce how the SPKI certificates are used to implement the higher level security policy, I describe why a certificate repository is needed and how the certificate repository is implemented for the TeSSA architecture.

Certificate Repository

SPKI assumes that the user who want to perform an action will provide for the validator all needed proofs, a set of certificates, that she has rights. In TeSSA, "the user" can be an inactive object, for example a small Java program, that cannot collect the needed proof by itself [67]. The certificates must be stored some easily reachable place where the validator can fetch

them. The TeSSA implementation uses Internet Domain Name System for storing the certificates.

Internet Domain Name System (DNS) is originally meant for connecting human readable Internet domain names to addresses of Internet Protocol, and thus making the use of Internet nicer for human users [62]. The DNS system is distributed database that makes fetching hierarchical names efficient. DNS also defines syntax for Internet names and how the namespace is distributed. The DNS system defines different types for objects that can be stored to its database, and RFC2539 [24] defines how to store certificates into the DNS database. The structure is same for all kinds of certificates, e.g. X.509, SPKI, and PGP.

In TeSSA architecture, SPKI certificates are stored to DNS servers based on the issuer and/or subject of the certificate [69]. Certificates that denotes absolute and direct trust by the issuer on the subject are the root of the trust and origin of certificate chains. Because this original issuer of trust is usually also the verificator of the certificate chain and the issuer knows the location of its own DNS server, the appropriate storing place is the DNS server of the issuer. Certificates giving a permission to a subject or certificates that confirm an identity of a subject are stored to the DNS server of the subject. It is the logical place for storage because the subject of the certificate knows its identity and its permissions. Delegation certificates are problematic because they can be seen as trust certificates or as permission certificates. Tuomas Aura found out that two way search algorithm is faster than others for searching delegation certificates [7]. For this reason, the delegation certificates should be stored to the DNS server of both issuer and subject [69].

The main problem of using DNS servers as a SPKI certificate storage is that DNS needs names for certificates: the anonymity of users may compromise. In the DNS resolver implemented for the TeSSA architecture [44], certificate names are based on the issuer or the subject of a certificate and a hash of the key to which the authorization is given. These are not the problem because user can create new keys and names for herself. The problem is the domain part of the name, which reveals where the user comes from. Still, it is needed, or the DNS server acting as a certificate repository cannot be found.

Trust and Policy Management

The TeSSA architecture uses Simple Public Key Infrastructure (SPKI) [28] for the more fine grade access control management than what IPsec can

provide and for binding the access control management to the operating system. SPKI itself is introduce in more details above, in section 4.4.

One purpose of the TeSSA architecture is access control and right management of software agents. The creator of an agent is seen as its authorizator and, finally, the operating system controls the computer hardware, i.e. which agents have rights to do what kinds of operations. The first certificate of a certificate chain must always be trustworthy [67]. This is truth even in traditional identity certificates though this is not always the reality (e.g. trust of some global company).

All the trust relationships in TeSSA are represented by SPKI certificates. In TeSSA, the certification loops are formed to both directions, both the authorization of an software agent and the authentication of a server or other node where the agent code is executed.

In the next section, the distributed trust and policy management of the TeSSA architecture is presented in more general level.

4.5.2 Four Kinds of Trust

The TeSSA Architecture is designed for expressing all kinds of trust relationships in distributed systems [67]. This include also other than access control like trust relationships. The purpose of TeSSA is to enforce a security policy. For this purpose, four kinds of trust exists in distributed agent systems [67]:

- **Execution:** Trust that the execution of program code is done loyally.
- **Naming:** Binding local names to cryptographic keys is a traditional form of trust, and it is done properly.
- **Authorization:** Trust that the access rights are not misused by the authorized party.
- **Delegation:** Trust that delegation is done properly, i.e. the issuer has the authorization it gives further.

In addition to access control type of trust, Nikander argues that security policy should state about all the four kinds of trusts [67]. Next subsection represents how a security policy works in the TeSSA architecture.

4.5.3 Security Policy

A security policy has two "levels": it can be organizational document that defines what confidentiality, integrity, and availability means, or it can give strict technical rule set for denoting trust. In TeSSA, the trust is described as SPKI certificates and like in PolicyMaker, the security policy decisions are based on authorizations, not identity, of an entity. Trusted third parties can only give recommendations and the security policy can define how trustworthy these recommendations are.

A security policy should define how it trust to so-called trusted third parties. In TeSSA architecture, the security policy controls how software agents can use resources on behalf of users. In such an agent system, the trusted third parties can be trusted for at least four purposes [67]: Authorization Authority gives access permissions, Agent Authorization Authority controls agent creation, Execution Environment Authority certifies trusted nodes, and Naming Authority gives names in applications in a secure manner. The next subsection describes these trusted third parties more precisely.

Trusted Third Parties

An Authorization Authority delegates access authorization on behalf of a resource. It can get the right directly from the resource or from another authorization authority. Like all trusted third parties, the authorization authority can only give recommencations. It is up to the security policy of the validating party if the certificates are accepted.

Agent Authorization Authority is special case of the Authorization Authority. It defines if an agent can be created and what kinds of rights the new agent has. For example, how much memory or network bandwidth the created agent can use.

Because agents may have different requirements about security, an Execution Environment Authority is needed. A network may have malicious host that try to benefit from agents, and to prevent this the trustworthy nodes must somehow be distinguished from other nodes. Other reason for this kind of TTP is that an agent may be divided to subagents which have different needs of security.

Human users are experienced to use names as identifiers. A Naming Authority will provide secure naming for utilization of users. Finally, the user is the entity which makes the decisions and a system is more secure if the user knows what she is doing. The Naming Authority is used to determine keys

and locations of a server.

Trusted third parties can only give recommendations for users and agents. A security policy defines how these recommendations are handled: are they believed or not.

Recommencations of TTPs

SPKI does not restrict the length of a certificate chain. This means that a source of authorization can only decide if it gives a right for further delegate the rights to a recipient of the certificate. The source cannot restrict to whom the recipient gives the rights and if it allows further delegation. Some certifier may authorize an entity which is completely untrustworthy from the viewpoint of the original source. In TeSSA, this is not a problem because the certificates are considered to be recommendations, not the final truth.

Finally, a source of authorization is usually the entity which checks that the certifate chain is valid and the requester is authorized to perform the requested action. A security policy of the source defines how trustworthy are the issuers of the certificates in the chain, and determine if the chain comply with the security policy. The security policy can be represented as SPKI certificates. For example, a security policy may define that reduction certificates are not trusted. Then the requester must present the original certificates instead of the reduction certificate.

4.5.4 Decentralized Management

The main purpose of the TeSSA architecture is to allow decentralized management of software agents. Each agent has its own rights and requirements for hosts, and each host has its own requirements for agents. Security policies and rights of the agents are declared as authorization certificates. Authorization certificates allow the management of such a system be decentralized because proofs of authorization can be taken along. In TeSSA, the credentials can also be stored in a repository which allows the agents to be small devices without large storage capability.

Decentralization of a system does not mean that it has no need for management. The management of the system is also decentralized. This means that each part of the system can have separate administration and different kind of security policy. The decentralized system is dynamic, and joining of new entity is relatively easy. New entities can be nodes or trusted third parties. Of course, the system may be used by new agents as well.

The main purpose of a node is to protect its local resources such as CPU or user data. For this task, the new node inserted to a decentralized system needs a key pair which is used to issue certificates. These certificates define to what trusted third party the node trusts, what kinds of recommendations it accepts, and what is the access control policy of the node [67]. The simplest case, two certificates are needed: The first certificate gives to an administrator all rights and right to delegate these rights. The second certificate defines the recommendation policy of the node, i.e. how the administrator can certify others.

The TeSSA architecture allows to easily add new trusted third parties to a decentralized system. Adding a new TTP does not change the policy of nodes because the nodes trust their addministrators to define how the new TTP is trusted. Of course, the trust to the new TTP must be based on something. Usually, several addministrators need to state that they trust to the new TTP before it can be accepted to be a system wide TTP.

In TeSSA, rights can be revoced, for example, if an agent misbehave and try to harm a node. TeSSA offers two ways to revoce the rights given in SPKI certificates. The first is normal SPKI certificate revocation implemented by online validity checking. A SPKI certificate gives in its validity field a location of a server which then verify if a certificate is still valid. The second way to revocate a right is restriction of security policy or recommendation policy. For example, if a trusted third party is no longer trusted, its recommendations can be defined to be untrustworthy.

This section has introduced five different kinds of authorization approaches. KeyNote is based on PolicyMaker, and SDSI and SPKI are merged. TeSSA uses SPKI certificates and build up a complete architecture for using them. Basically, the solutions can divided to two different ones: PolicyMaker kind of solution and SPKI kind of solution. PolicyMaker, KeyNote, SDSI/SPKI, and TeSSA are compared according to the comparison methodology in the chapter 6. The comparison methodology is introduced in the next chapter.

Chapter 5

Comparison Methodology

Deborah Hix and Robert Schulman introduced a comparison methodology for human-computer interface development tools [46]. They also give characteristics for a good comparison methodology: A good comparison methodology will use standardized approach to get reliable and quantifiable results. It should be objective, easy to use and adaptive, i.e. if a new tool is introduced, it can be evaluated equally to old ones with the same comparison methodology. Comparison of different kinds of solutions should always be independent from external interference. This means that, for example, software and hardware of underlying system should be same for every solution.

Donald Norman argues that proper way to design things is the human centered way. Computers as an infrastructure should be invisible for people that are communicating through them. He also states that technology is relatively easy to change, the human behavior based on culture and social connections build up during hundreds of years changes slowly [70]. Jacob Nielsen argues that user interfaces has become more important when a personal computer is available for wider audience and computers has become common even in homes [66]. Thus, the main focus of the comparison in this thesis is usability.

This chapter presents the comparison methodology used to evaluate different decentralized authorization methods and systems. First, I introduce shortly Common Criteria (CC) which is used to evaluate security properties of a system. In the second section I describe how heuristic evaluation, the chosen usability evaluation method, is used to evaluate the properties identified by Common Criteria. In the last section, I describe different kind of using situations which will act as test cases in the actual evaluation. The next chapter gives the results of my evaluation.

5.1 Common Criteria

Since 1985, several security criteria has been published to help governments to choose secure products. In 1995, all the criteria were merged to be one unified standard called Common Criteria (CC) [81]. Common Criteria defines requirements for security properties of information technology products and systems, and provides a criteria for comparison [17]. The security properties evaluated in CC are confidentiality, integrity, and availability of a system. Security requirements are divided in classes, for example, identification and authentication is one class. Members of a class are families, which define set of security requirements. Each such requirements is called a component.

Common Criteria as such does not suites well to compare different decentralized authorization mechanisms. I use it to define the functional parts of the systems and to identify usability requirements for those parts.

The next subsection presents the CC classes and families which can be found from the decentralized authorization systems presented in chapter 4.

5.1.1 CC Classes and Families

Common Criteria defines eleven classes of security functional components. Six of them suites for dividing decentralized authorization mechanisms to components.

Cryptographic Components

CC Class FCS defines cryptographic components. Decentralized authorization systems have both of CC cryptographic components: FCS_CKM Cryptographic Key Management and FCS_COP Cryptographic Operations. The first component family consists of key generation, key distribution, key access and key destruction. The other family, cryptographic operations, consists of all typical cryptographic operations such as encryption and decryption, signing and verification, calculation of cryptographic checksums and secure hashes, and key agreement. CC defines, that these operations must be performed according to given algorithms and with correct key sizes.

User Data Protection

Next suitable CC Class is FDP User data protection. The first family in this class is FDP_ACC which defines Access control policy. The access control policy has three traditional parts: subject, objects and operations. Security attributes used by the access control policy are defined by FDP_ACF Access control functions.

Common Criteria defines other family for Information flow control policy and functions (FDP_IFC and FDP_IFF) similarly to access control policy for defining data transmissions. FDP_ITC defines mechanisms for importing user data from outside of the evaluated target. In a decentralized authorization system this is needed, for example, when a user collects necessary credentials from trusted third parties. Other family, FDP_ITT Internal transfer, defines protection for data transferred inside a system.

Other family, FDP_SDI Stored data integrity specifies that data in a system does not change during storage or internal transmissions. Also confidentiality can be ensured inside a system (FDP_UCT Internal user data confidentiality transfer protection). For a trusted system, the integrity is usually more crucial than confidentiality. Similar family protects data from modifications during transmission between systems (FDP_UIT Inter system user data integrity transfer protection). For integrity, the FDP class provides data authentication by signature (FDP_DAU data authentication).

Identification and Authentication

Common Criteria class FIA Identification and authentication defines functions to establish and verify user identity. It does not define authorization, but still this class can be used as foundation when the functions of authorization are determined.

Users of a system have security attributes. FIA_ATD user attribute definition CC family defines how the attributes of a user are associated with the user. In decentralized authorization system the security attributes are binded to a user presented by a public key with certificates, i.e. the attributes are not bound to the identity of a user like the Common Criteria defines.

FIA_UAU User authentication family defines mechanisms for authenticating users and gives required attributes for the mechanism. Similarly, FIA_UID User identification defines conditions for identity mechanisms. For example, both can define that the authentication/identification must be done before every action.

FIA_USB User-subject binding associates security attributes of a user with other entity that acts on behalf of the user. This is

Security Management

Common Criteria defines a class for management of security aspects such as functions, attributes, and data in a system. Respectively, the families are FMT_MOF Management of functions, FMT_MSA Management of security attributes, and FMT_MTD Management of data. First family defines that authorized users can manage functions etc.

Security management has also other aspect. Authorized entities can also decide about revocation with FMT_REV Revocation family. Security attributes can have validity limitations given by FMT_SAE Security attribute expiration.

CC family FMT_SMR Security management roles determines what capabilities are given to a role and which users have the role.

Privacy

Privacy class gives requirements for functions that protect a user of identity discovery and misuse. FPR_ANO Anonymity family gives a user a possibility to use a service anonymously.

Other way to protect identity disclosure is use pseudonymity (FPR_PSE). It allows a user be anonymous but still be accountable. This means that the identity of a user can be revealed by authorized user if, for example, the user performs an illegal action.

One part of privacy is untracebility of actions of a user, i.e. when a user uses several services, these cannot be linked together. CC notices this by defining FPR_UNL Unlinkability family of functions.

Interestingly, CC defines FPR_UNO Unobservability family: a user can use a service without being observed. Of course, the actions that are allowed to be performed without observation must be defined. There may also be users that are authorized to observe usage of some services.

Communication

Parts of a decentralized system need a secure way to communicate with each others and with users. CC Trusted path/channels class defines families to

provide trusted path between parts of a system.

Common Criteria defines two families of trusted channels: FTP_TRP Trusted path defines a communication path from a user to a system and FTP_ITC Trusted channel defines a communication path between trusted systems. Both families give requirements for establishment and maintenance of a connection.

5.1.2 Elements of Decentralized Authorization Systems

The parts of common decentralized authorization system are identified in this section with help of Common Criteria. I will not evaluate their functionality from the technical point of view. I use them as a basement for defining usability test tasks of different user groups. The common functional parts of decentralized authorization systems are listed below.

- Key management
- Cryptographic operations
- Access control policy
- Handling of credentials
- Security attribute management
- Authorization
- Delegation
- Management of credentials
- Revocation
- Validity
- Roles
- Privacy
- Logging
- Trusted communication

Some of the listed functional parts are out of the scope of this thesis. For example, I do not evaluate the usability of cryptographic operations and trusted communication paths used in decentralized authorization mechanisms. Elements that are essential and characteristic particularly for decentalized authorization systems are handling of credentials, security attribute management, authorization, delegation, management of credentials, revocation, validity, roles, privacy, and trusted communication.

5.2 Usability Test Method

Comparisons should always be unbiased. All the decentralized authorization systems presented in chapter 4 approach the problem of decentralized authorization from different point of views. Two are systems based on same kind of architecture, two are merged to be a authorization certificate system, and one is an architecture constructed from standard components. Complete implementations of all the systems are not available because the work is unfinished or because of US exportation laws. For this reason, I evaluate usability and understandability of the solutions from different user point of views without actual implementations. This limits the selection of usability test method.

5.2.1 Choosing Usability Test Method

The section 3.1.3 above presents several usability evaluation methods. Jenny Preece *et al.* gives issues that should be considered when selecting evaluation method [74]:

- Purpose of the evaluation
- Stage of the system
- Involvement of users
- Qualitative or quantitative data
- Practical considerations

I have given above one practical consideration for my evaluation: the lack of implementations. For comparing designs, experiments and benchmarking methods are best but they require something concrete to test. Some of the

Method name	*Users needed*	*Main advantage*	*Main disadvantage*
Heuristic evaluation	none	Finds individual usability problems.	Does not show surprising problems in users needs.
Thinking aloud	3-5	Point out user misconceptions.	Unnatural and difficult for users.
Observation	3 or more	Reveals real tasks of users.	No control.

Table 5.1: Usability method comparison [66]

targets are in early state which should be compared with predictive methods such as heuristic evaluation or observation. Jacob Nielsen has also compared the different usability methods. He states that if only few users are available for usability tests, the chosen method should be heuristic evaluation, thinking aloud, or observation [66]. Table 5.1 summarizes Nielsen's comparison of these three methods.

None of the usability evaluation systems are good for evaluation of concepts or architectures, i.e. solutions without user interfaces. In this thesis, I apply mainly heuristic analysis but use also cognitive walkthroughs to compare the decentralized authorization systems presented in previous chapter. The using of more than one method for evaluation helps to gain more reliable results.

The heuristic analysis and cognitive walkthroughs usability method are introduced in this section. The end of this section I describe how to apply these methods to evaluation of decentralized authorization systems.

5.2.2 Heuristic Evaluation for Decentralized Authorization

Heuristic evaluation is developed by Rolf Molich and Jacob Nielsen [63]. The purpose of the evaluation is to find usability problems of a developed system in an iterative process. However, the evaluator is not actually using the system. This means that the user interface of the system does not need to be finished and the heuristic evaluation can be used in any phase of the design, even very early.

In heuristic evaluation, small set of evaluators examine the system and compare it to usability principles. Jacob Nielsen argues that "heuristic evaluation was originally developed as a usability engineering method for evaluators who

had some knowledge of usability principles but were not necessarily usability experts as such" [65]. Of course, the experience of the evaluators affect to how many problems they may find from the system. For getting reliable results, about five evaluators should be used [66].

Usability Principles

For performing a heuristic evaluation, Nielsen and Molich present a ten-point check-list of generic usability principles, against which the system's usability can then be cevaluated and most common usability problems can be detected [63]. Based on this work Nielsen has developed following list of usability principles for user interface designers [66]:

- **Simple and natural dialogue** means that no irrelevant or rarely used information are presented in the dialog, and the information is presented in logical order.
- **Speak the users' language** and do not use unfamiliar technical slang.
- **Minimize the users' memory load**, i.e. do not require the user to remember information from other dialogs. Also, instructions or other help should be easily accessible.
- **Consistency** means that same thing is always called with same name/word.
- **Feedback** means that user should always be able to follow what is going on in reasonable time. Response time less than one second does not need indication what the system is doing.
- **Clearly marked exits**. A function can be chosen by mistake and thus the user should have easy escape from every function.
- **Shortcuts** make the system usable also for experienced user who does not need guidance. The shortcuts should be invisible for the novice user because she may be confused by them.
- **Good error messages** are preventive, constructive and informative. They do not use error codes but describe with words the problem. An error message should also suggest what the user should do next.
- **Prevent errors** beforehand is better approach than the previous principle of providing good error message.

- **Help and documentation** which allow easily to search problems and give concrete steps to solve them. It is better if the system is usable without help and documentation, but sometimes it is necessary.

Evaluation Procedure

In heuristic evaluation, the evaluator inspects each parts of the user interface from dialog to dialog. Each dialog or screen is compared to the usability principles (heuristics). The used heuristic will effect to the findings of the evaluators but the method is systematic.

Nielsen has categorized usability problems that can be found with heuristic evaluation [65]: The first category of usability problems affect only to a single user interface element such as dialog-box, error message, or menu. The second category consists problems that require comparison between elements. For example, each dialog is fine but when comparing the dialogs, each use different word to denote quitting: exit, quit, close etc. Structural problems of dialogs are the third category. For example, navigation through large menu structure should be symmetric. The last category user interface problems are missing elements that should be in a dialog. These categories can also be helpful when evaluating a system because they help the evaluator to concentrate to find problems of different layers.

5.2.3 Cognitive Walkthrougs for Decentralized Authorization

Like heuristic evaluation, cognitive walkthroughs is a kind of structured expert reviewing method. In cognitive walkthroughs, the system is analyzed and basic tasks of the users are described as scenarios. Each scenario describes who will try to perform the task and what she knows, what goals and subgoals the user has and what is her motivation. The scenario also describes the tools and resources the user has and how the user can use the system [74].

When the scenarios are identified and described, the actual cognitive walkthrough can be performed. In the analysis, a user carries through the scenario subtask by subtask. The evaluator observes user and what she do in order to accomplish the tasks. Usability problems are marked up for later to be fixed.

5.2.4 Applying Heuristic Evaluation and Cognitive Walkthrougs to Decentralized Authorization

Whitten and Tygar have tested usability of the user interface of an identity public key infrastructure, i.e the usability of PGP 5.0 [82]. Their test was based on cognitive walkthroughs and usability tests. Because there is no working user interfaces for all the decentralized authorization systems, I use first part of cognitive walkthroughs to describe main tasks of the different user groups and heuristic evaluation to identify the usability problems of the systems. The main tasks correspond the elements identified by CC above in section 5.1.2.

Whitten and Tygar end up to define that a security system is usable if its users are aware of the security tasks, know how to perform the tasks, do not make fatal errors, and are comfortable with using the system [83]. Similarly, my heuristics is

- the user can know what she is doing,
- the user can find out how to perform the action, and
- the possibility of perform an error is minimized.

5.3 The Test Problems

The purpose of this thesis is to compare usability of the different decentralized authorization approaches. The comparison does not compare the actual user interfaces but the concepts and the architecture of the approaches with methodology presented in the above section.

The section 3.1.2 introduces shortly basic goals and tasks of different user groups of a decentralized authorization system. Different elements of the system are identified with help of Common Criteria in section 5.1.2. Based upon the goals, the tasks and the elements, following test cases are identified.

For test cases, the users are divided to three groups: administrators, developers, and common users. The administrators and developers need to be familiar with programming and, at least, traditional public key infrastructures. The programming skills of administrators can be more modest than the developers.

In this section, the test cases are introduced according to cognitive walkthroughs phase one. The actor, the goal, and the motivation are given in

this section. The description how the goal can be archieved is given in the next chapter while comparing the systems because it is dependent on the system.

5.3.1 Test Cases

Case 1: Authorization of Entities

Who: Administrator

What and Why: The actual source of authorization is the resource itself but usually it delegates the full authorization to the administrator of the resource. Thus, the administrator can be seen as the source of the authorization. Administrators tasks is to delegate different authorizations to other entities. It must be sure that the entity is trustworthy to have the authorization.

With: A decentralized authorization system

How: Depending on the system

Case 2: Definition of Security Policy for a Node

Who: Administrator

What and Why: A security policy defines what kind of authorization is needed for using the different resources of the node. The node relies that the authorized users behave properly and according to the rules defined by the policy.

With: A decentralized authorization system

How: Depending on the system, but the systems have similarities due to a common procedure of defining a security policy. Firstly, the upper layer security policy is defined by directors of an organization. The security policy of a node in the organization is derived from this upper layer policy. Writing the policy for the node according to the used authorization system is the task of the administrator.

Case 3: Definition of Security Policy for a Piece of Code

Who: Developer

What and Why: A security policy defines what kind of autorizations the piece of code requires from a node before it can be executed in the node. It can also define what resources the piece of code needs in order to be properly executed in the node. Similarly than defining security policy for a node, the developer can also define who can use the piece of code.

With: a decentralized authorization system

How: Depending on the system

Case 4: Handling of Certificates

Who: Administrator, Developer, User

What and Why: Authorization information is stored in a certificate. The certificates are signed, which protects the integrity of the certificate. The certificates must be stored somewhere. Some certificates are handled by the end users, others by developers, and most of the certificates are handled by the administrators. Each of the user groups needs to know, when a certificate is needed and to whom it must be send, and through what kind of channel.

With: A decentralized authorization system

How: Depending on the system

Case 5: Revocation of Certificates

Who: Administrator

What and Why: Sometimes certificates become unvalid during their lifetime. In some systems, the certificates can be revocated. This task belongs to the administrator of the system.

With: A decentralized authorization system

How: Depending on the system

Case 6: Checking Validity of a Set of Certificates

Who: Administrator, Developer

What and Why: Checking that a set of certificates is valid has two parts. First, each certificate of the set must be valid. The digital signature of the certificate must be correct and done by the issuer of the certificate. In addition, the certificates have often other validity conditions such as time

limit or that the certificate is not revoked (case 5). The issuer of the certificate defines how the validity of the certificate must be checked. Usually, the validity is asked from the issuer itself, but the issuer can also delegate the validation of certificates to others.

Secondly, the set of certificates must form complete proof, i.e. certificate chain from the owner of the resource to the requesting entity. Certificate chain is presented in section 2.3.1.

With: A decentralized authorization system

How: Depending on the system

Case 7: Privacy of Users

Who: User, Administrator

What and Why: Authorization systems support privacy of the user because the user identity is not essential for the authorization. But still, the identity of the user can be revealed if information gathered from several credenentials are combined. In order to protect privacy, only credentails necessary for gaining authority should be used.

With: A decentralized authorization system

How: Depending on the system

Case 8:Distinguishing Trusted Channels from Untrusted Channels

Who: Administrator

What and Why: In addition to signed certificates, systems may use internal certificate-like structures to denote policies. Usually these unsigned policies are handled internally, but if they are send to some other fully trusted entity, the transmission path must be trusted and secure. The task of the administrator is to define when to use untrusted or trusted channel.

With: A decentralized authorization system

How: Depending on the system

In the next chapter, these test cases are completed for each compared decentralized authorization system and used as basis for heuristic evaluation of the systems.

Chapter 6

Comparison and Evaluation

In this chapter, different decentralized authorization approaches presented in chapter 4, namely PolicyMaker, KeyNote, SDSI/SPKI, and TeSSA, are compared according to the comparison methodology introduced in chapter 5. The main objective is to find out which of the systems is the easiest to understand and use in different kinds of situations without making errors.

6.1 Tests

Section 5.3.1 introduces eight test cases which presents common operations of different user groups of decentralized authorization systems. This section search for usability problems in the four systems based on those test cases. The main purpose is to evaluate if the user can know what she is doing (e.g. what knowledge is required from the user), can she find out how to proceed in order to perform actions, and can she accidentally make a fatal error. Each test case is discussed separately, followed by a summary that gives the differences of the decentralized authorization systems.

6.1.1 Case 1: Authorization of Entities

The main purpose of decentralized authorization systems is to authorize entities to perform actions. The authorizations can also be further delegated. This section describes how entities are authorized in different systems and what usability problems there exist.

PolicyMaker

How

An administrator of a resource has received full or partial authorization from the resource or from another administrator, e.g. trusted third party. Before writing certificates, the administrator has somehow to be affirmed that the entity is trustworthy to gain the authorization. As described in section 4.1.3, PolicyMaker denotes authorization with certificates of form

Source **ASSERTS** *AuthorityStruct* **WHERE** *Filter.*

The administrator writes assertions of PolicyMaker with a text editor. She fills in her public key as the source of authorization, the authority structure denoting who is authorized and the optional filter which restricts the authority. Then she digitally signs the certificate with her private key.

The assertion can authorize a keyholder presented by a public key or more complex conditional clause as in the example presented in the subsection 4.1.4 where "three keys from a group of keys" has the authorization. The more complex authority structures are written with either AWKWARD programming language or regexp.

The filter of a certificate has two parts: the filter name and a program presented with AWKWARD or regexp. The administrator has to choose a suitable filter name for the condition. Possibilities are annotator, predicate, commentary, and application. When someone wants to perform an action, the application sends a query that contains proofs, i.e. certificates, and policy assertions of the application to the PolicyMaker compliance checker. Each proof corresponds a filter name of a policy assertion and the proof presented by the certificate has to be equal with the regexp or the AWKWARD program of the corresponding policy.

Evaluation

PolicyMaker does not give instructions how the administrator can gain the knowledge that an entity is trustworthy. Thus, the administrator has herself to decide when she authorize the entity.

When the administrator is ensured that the entity can have an authorization, creating a simple certificate for the entity is easy. Usually, the entity is presented by her public key. The only problem in writing the simplest certificate is that the administrator can by accident give authorization to a wrong entity because the keys are hard to recognize. Keys are long sequences of letters and digits, and every character is meaningful.

The little more complex form of certificates gives authorization to a group of public keys, e.g. at least three keys certified by a certain entity. This kind of certificate has the same problem of identifying keys as the simplest assertion, i.e. the key of that certain entity who has certified the other keys must be correct. Other problem is the writing of the authority structure with either an AWKWARD programming language or regexps. If the administrator is unfamiliar with programming, she can make mistakes and give authorization, for example, for a too small group of entities.

Writing a filter program for any assertion (certificates or policy assertion) is harder because the program must be compatible with queries written possibly by somebody else. Even using the same language for writing the policy assertions and the certificates does not help because the administrators must agree with others what the statements means.

PolicyMaker assertions are monotonous. This means that each assertion can only extends rights given to an entity, and thus leaving a certificate out from the proof of compliance does not give larger rights than giving all certificates. The administrator must be careful when defining the rights in the certificate. Otherwise, she can accidentally give excessively wide rights to the entity.

The delegation rights are not clearly separated in the certificates. When the administrator creates a certificate, the delegation right must be written together with the authority to the filter field of the certificate. PolicyMaker does not define what kind of delegation is allowed, and only the application knows what the filter means. The administrator may accidentally give right to further delegation of the right to an entity.

KeyNote

How

Basically, KeyNote works similarly as PolicyMaker. An administrator can issue assertions that denote authorization. KeyNote assertions have an authorizer, a receiver of the authority called licensee, and conditions for the authority. Of course, the assertions are signed by the authorizer.

The assertions can have local constants. For example, the administrator can give aliases for public keys. These aliases can then be used in the assertion. For example, the authorizer is usually a public key. This public key can have alias that can be used instead of the public key in the certificate.

The licensee can be one principal or several principals with conditions. Conditions can be “both of the principals A and B”, “some of the principals A

and B", or "k of the following set of principals". The licensees are expressed with regexps and aliases can be used instead of the public keys.

Conditions are simple test - value pairs. Tests can compare strings or numbers. The value belongs to a set that defines possible responses of the policy.

Evaluation

Like PolicyMaker, KeyNote does not give instructions how the administrator can be sure that licensees are trustworthy to get the authorization.

The administrator can give aliases for the keys used in an assertion which makes the writing of the assertion easier because the it is not necessary to write the keys several times. But if the administrator issue several certificates, she can accidentally try to use aliases of other assertions in the assertion. Wrong entity may gain access because two keys have same nickname. On the other hand, the aliases prevent from choosing wrong key.

KeyNote uses also common language for policies and certificates. Similarly to PolicyMaker, the semantics of the application is not understood by the compliance checker of KeyNote. Thus, the administrators must agree on what the semantics used by an application means.

KeyNote certificates are also monotonous, and the administrator must be careful that she do not issue to wide authorizations.

KeyNote certificates do not give clear separation if the right can be further delegated or not. The delegation right is written in the condition field.

SDSI/SPKI

How

SPKI certificates are quite similar to the KeyNote assertions. The SPKI certificate has issuer, subject, given right, and delegation fields. Of course, the certificates are signed by the issuer.

The issuer of the certificate is the public key of the administrator or a hash of the public key. The subject, which is the receiver of the authorization, can also be a public key or a hash of a public key but it can also be a relative or fully-qualified name, a certain group, or a hash of a piece of code.

SPKI certificates have a separate field for delegation. The administrator can give delegation rights to the receiver of the certificate. She cannot restrict the depth of the delegation, but she can restrict the given right.

The permission given by the certificate is defined with s-expression. Full

rights are given, if the permission field contains a character "*". The rights can only be reduced from that.

Evaluation

SPKI does not define how the administrator can be certain that an authorization can be given to an entity.

SPKI certificates usually uses keys or hash of keys as issuers and as receivers of the rights. Thus, the administrator who issues the certificate can accidentally give wrong key as the receiver of the right.

Someone has issued a certificate to the administrator and given her right to further delegate the authority defined by the certificate. The administrator can only delegate the right she has or she can restrict the right. She cannot accidentally extend the right or the certificate chain become unvalid.

TeSSA

How

TeSSA uses SPKI certificates for trust management.

The difference between TeSSA and SPKI in trust management is that TeSSA gives instructions how trust can be created [50, 51]. The trust can be formed based on rumors from other parties, e.g. friends and colleagues that are trustworthy. If someone else has already trusting to an entity, the entity is more trusted than another one that is not trusted by anyone.

Evaluation

Same problems presented earlier with SPKI certificates be materialized also with the TeSSA architecture with one exemption.

According to TeSSA, the administrator can collect information about trustworthiness of TTPs from other administrators.

Summary of Usability Problems of Case 1

Summary of the usability problems of the test case 1 in the different decentralized authorization systems is given in table 6.1.

Usability problem	PolicyMaker	KeyNote	SPKI	TeSSA
Gaining assurance of trustworthiness not defined	x	x	x	-
Choosing the correct keys	x	-	x	x
Need programming skills	x	-	-	-
Agreement with other what statements mean	x	x	-	-
Too much rights given by accident	x	x	-	-
Confusion with local names/aliases	-	x	-	-
Delegation right can be given accidentally	x	x	-	-

Table 6.1: Summary of Usability Problems of Case 1

6.1.2 Case 2: Definition of Security Policy for a Node

Writing rules for protecting information, resources or services of a node is probably the oldest form of security policy. Traditionally, the security policy is implemented as an access control list, but the decentralized authorization systems can also be used in defining security policies for nodes. The task belongs to the administrator of the node. Let us assume that the administrator knows what a security policy is. She knows what she is doing when she creates a security policy for the node: she is limiting access to the resources of the node.

This subsection presents how defining the security policy for a node can be done with different authorization systems, and discusses what kinds of difficulties the users may have while completing the task.

PolicyMaker

How

PolicyMaker use same language for certificates and policies. This means that like authorization of entities, the administrator writes policy assertions of PolicyMaker with a text editor according to the same structure as certificates. The only difference is that the source of policy assertion is marked with the word "POLICY", not a public key of the source. The policy assertions are not digitally signed because they are used only internally.

The policy assertion can also contains filters. When a query is evaluated that it complies with the policy, the filter name refers to a proof given in the query. The proof has to be equal with the regexp or the AWKWARD program of the policy.

Evaluation

The administrator has two variables to use: the authority structure can contain a public key or a structure presenting limitations for a group of public keys i.e. the receiver of the authorization, and the policy assertion can include filters, i.e. the restrictions for the authorization.

The security policy of an organization can be quite general. The administrator must decide what kind of policy assertion describes the security policy of the organization in the node. She has to identify information, resources and services of the node that needs to be protected, and define security policy assertions for the protection.

Same problems that arise in authorization of entities apply also to the writing of policy assertions because the certificates and the policy assertions of PolicyMaker uses same syntax for authorizing entities.

Policy assertions are not signed. Afterwards, if integrity and reliability of a system is compromised, there is no way to distinguish integral policy assertions from falsified assertions.

Humans are often considered as the main security thread, and even trusted personnel can make mistakes by accident or on purpose. Because all policies are issued by the POLICY, there is no way to distinguish afterwards who has issued the policy assertion, i.e. which administrator needs more education or to be fired.

PolicyMaker does not offer possibility to make sure that a policy assertions fulfill a part of the security policy. However, testing of individual policy assertion is possible. The more difficult task is to make sure that a set of policy assertions fully covers the security policy.

KeyNote

How

Similarly to PolicyMaker, KeyNote uses same language to denote policies and certificates, and the word "POLICY" as the authorizer of a policy assertion.

In addition to policy assertions, the administrator of the node must define what kinds of access rights can be given, e.g. no access, limited access, and

full access. This set of possible rights must be organized from the lowest to the highest right.

Evaluation

KeyNote does not give instructions how a security policy of an organization can be written to policy assertions. But because of the set of possible values, the creation of policy assertions may be easier. These values can be derived from the security policy and they may help forming the policy assertions with KeyNote.

The set of possible compliance values must be ordered from the minimal trust to the maximal trust value. If the possible compliance values derived from the security policy varies much from their content, their ordering may be difficult for the administrator.

Even through the policy assertions and compliance value set will help the administrator to define the policy, the administrator may fail in defining the resources that needs to be protected.

KeyNote inherits the liability problems from the PolicyMaker. The reserved word POLICY does not tell the actual issuer of the policy assertion. Similarly, because policy assertions are not signed in KeyNote, their integrity is not guaranteed if the storage is compromised.

Because KeyNote uses same syntax for authorizing entities and defining security policies, the same problems than presented in subsection 6.1.1 arise in writing of policy assertions.

SDSI/SPKI

How

The security policy of a node is defined as common SPKI certificates, i.e. there is no difference between certificates presenting a security policy and other certificates.

The resource, a node in this case, itself usually issues the first certificate of a certificate chain. The issuer is marked with a reserved word "self". However, this certificate is signed by so-called "self key" of the resource. The certificate usually gives all rights and right to further delegate to the administrator of the resource.

Evaluation

The administrator issues SPKI certificates that describes the policy of the

Usability problem	PolicyMaker	KeyNote	SPKI	TeSSA
Expressing the policy of organization as certificates	x	(x)	x	x
Identifying resources that needs protection	x	x	x	x
Defining possible compliance values as ordered set	-	x	-	-
Integrity of policy certificates unprotected	x	x	-	-
Liability of administrators inseparable	x	x	-	-
Verifying that a set of certificates corresponds to a security policy	x	x	x	x

Table 6.2: Summary of Usability Problems of Case 2

node. SPKI does not define how the authorizations corresponds the rules of the security policy and it does not help to identify the resources that have to be protected.

TeSSA

How

TeSSA uses SPKI certificates to describe security policy for a node.

Evaluation

Usability problems in the TeSSA architecture are similar as when using SPKI certificates.

Summary of Usability Problems of Case 2

Summary of the usability problems of authorizing nodes in the different decentralized authorization systems is given in table 6.2.

6.1.3 Case 3: Definition of Security Policy for a Piece of Code

PolicyMaker

How

Defining a security policy for applications is the the original purpose of PolicyMaker. Of course, the piece of code can be an application.

PolicyMaker works similarly when authorizating a piece of code as in authorizating a node: the developer of the piece of code writes policy assertions that describ the security policy of the piece of code or the program.

Evaluation

Authorization of a piece of code has also other usability problems than those introduced in authorizing an entity and a node. The developer (or the administrator) needs to know who can gain the right to use the program and with what conditions, which nodes are trustworthy execution environments, and finally, what kinds of rights the program needs in order to work correctly.

The developer of the program can define trustworthy third parties to point out the trustworthy nodes. Creating this kind of policy assertion has similar problems as presented in the previous cases, i.e. knowing the right keys, knowing the trustworthy parties etc. Similarly, the usability problems of the security policy that defines who has the right to execute the program are given in the previous cases.

What makes the definition of security policy for a piece of code different from defining other security policies is that the piece of code can "move" from node to node and be executed in different kinds of environments. The developer of the program knows what resources her program needs, but she cannot know which node will offer these resources.

KeyNote

How

In Keynote, creating a security policy for a piece of code does not differ from the creation of policy for a node.

Evaluation

Usability problems in defining KeyNote policy assertions for a piece of code are similar to those presented in the cases one and two.

KeyNote does not help of finding the resources which the piece of software may need.

SDSI/SPKI

How

SPKI does not distinguish the creation of security policy from other delegation of authorization. Thus, there is no difference to whom the certificate is issued or for what purpose the certificate is created.

Evaluation

Because the issuing of certificates that defines the security policy of a piece of code does not differ from issuing other certificates, the same usability problems occurs.

Like PolicyMaker and KeyNote, SPKI does not help locating needed resources. The administrator must find out which trusted nodes provides the resources.

TeSSA

How

The original purpose of the TeSSA architecture is to provide trust management for a distributed agent system. The agent is a piece of code that needs to trust the execution environment of a node and authorization to use resources of the node.

Pasi Eronen *et al.* has extended the Jini architecture to use decentralized trust management [33]. The Jini architecture provides decentralized location service where the nodes can be registered. This helps to find out suitable secure node for execution of the piece of code.

Evaluation

However, TeSSA itself does not provide solution for locating the nodes who will provide the needed resources and can guarantee the access to those resources.

Usability problem	PolicyMaker	KeyNote	SPKI	TeSSA
Location of needed resources	x	x	x	(x)

Table 6.3: Summary of Usability Problems of Case 3

Summary of Usability Problems of Case 3

Summary of the usability problems of authorizing a piece of code is given in table 6.3. The main usability problem is the finding suitable nodes where the piece of code can be executed. Probably, this is not even the task of a decentralized trust management system.

6.1.4 Case 4: Handling of Certificates

Certificates can be stored in different party of a decentralized authorization system. When someone want to perform an action, the certificates must be collected and presented as a proof that the entity has authorization to do what she wants to. In this section, I describe who is responsible of storing the certificates and who collects them when the certificates are needed.

PolicyMaker

How

In PolicyMaker, the application collects all needed certificates on behalf of the user from trusted third parties. After all, the application is the entity who has protected resources in the PolicyMaker architecture. When the PolicyMaker is used in wider context, the entity that has protected resources, e.g. a node, will collect the certificates and sends them with the query to the compliance checker of PolicyMaker.

In the PolicyMaker architecture, two types of credentials are added to a query: the local security policy of the application and certificates issued to the requesting entity.

First, the administrator of the application defines trusted third parties that can provide certificates for end users. She creates certificates that delegates the right to further delegate the access to the application to trusted third parties. The administrator can limit the rights given to the TTPs. Creating certificates are introduced in previous cases. These certificates are send to the compliance checker with a query.

The administrator has to configure the application to request from all trusted third parties the certificates that may be suitable for a user who requested an action. The TTPs will send the certificates to the application that forwards them with other certificates and policies to the compliance checker. The compliance checker checks that the requested action and the given proofs, i.e. certificates, complies with the policy. The compliance checker of PolicyMaker can also give hints what kinds of certificates are needed in addition to the given certificates before the request can be approved.

Before the certificates are sent to the compliance checker, the application has to check that they are valid, i.e. the signatures are correct, validity time has not exceeded etc. This is discussed later in this section.

Evaluation

The problems of creating certificates and creating policy assertions apply also in this case.

The administrator of the application must somehow be sure to which third parties it can trust to further delegate a right. She must create a kind of list to define these TTPs which are used as repositories of credentials, i.e. she must form policy assertions that defines TTPs and their authorities. The list may not be complete because some TTPs can be trusted to collect the needed certificates from the entities that they trusts. When a user sends a query for using the application, the certificates are needed and the application must fetch the certificates from the trusted third parties.

In addition, the administrator of the application must check that all the certificates and policies are valid before she sends them to the PolicyMaker compliance checker.

KeyNote

How

Similarly than PolicyMaker, the application or other protected resource will collect the needed certificates and policies. KeyNote does not define that the certificates are collected from trusted third parties

The main difference between PolicyMaker and KeyNote is that in KeyNote the checking that the certificates are valid can be done either by the protected resource, e.g. an application, or by the KeyNote compliance checker. PolicyMaker requires the application to verify certificates, and the PolicyMaker compliance checker only verifiest that the certificates proofs that requested

action is allowed. If the compliance checker is used to verify the certificates, a cryptographic algorithm that is known to the KeyNote must be used.

Evaluation

If KeyNote compliance checker is used to verify the certificates, the administrator must be sure that the certificates use known cryptographic algorithms. Otherwise, the KeyNote will assume that the certificate is part of the security policy, i.e. it is an unsigned policy assertion.

SDSI/SPKI

How

In SPKI, the requesting entity will provide all the needed certificates when she makes a query to use a resource. The owner of the resource then checks if the certificates form a valid chain that gives authorization to use the system to the requesting entity.

Evaluation

If an entity that needs authorization to use a system has small memory capacity, it cannot store many certificates. Thus, when the administrator creates certificates for such entity, the certificate chain must be as short as possible. This limits the possibilities to use certificates created by trusted third parties. It is possible to use reduction certificate that compress a chain of certificates to one certificates, but these certificate reduction certificates may not be approved by the policy of the resource.

TeSSA

How

In the implementation of the TeSSA architecture, SPKI certificates are stored either in a local repository or in the DNS system. The entity who want to use a resource does not need to provide all the credentials. The owner of the resource can deduce from the given certificates where to find the other needed certificates.

Evaluation

The administrator of the protected resource must deduce if the certificate chain needs to be completed by fetching additional certificates.

Usability problem	PolicyMaker	KeyNote	SPKI	TeSSA
TTPs must be listed beforehand	x	-	-	-
Protected resource collects the certificates	x	x	-	x
Requesting entity collects the certificates	-	-	x	-
Protected resource checks the validity of certificates	x	(x)	x	x

Table 6.4: Summary of Usability Problems of Case 4

Summary of Usability Problems of Case 4

Summary of the usability problems of the collecting certificates in the different decentralized authorization systems is given in table 6.4. The entity that collects the needed certificates may cause problems to the administrators regardless what this entity is.

6.1.5 Case 5: Revocation of Certificates

A private key of an entity may be revealed to an outsider which makes misuse of certificates issued to the entity possible. In some systems, the certificates can be revoked by the issuer of the authorization. This section discusses what usability problems there are in the revocation of certificates in different systems.

PolicyMaker

How

PolicyMaker itself does not define revocation services [11] but revocation of certificates is possible to organize. A certificate can contains a public key of a certificate revocation service that reports which public keys are non-revoked.

The possibility to revocate a certificate must be written in the certificate when it is created. Thus, the administrator who issues the certificate must write more complex certificates: instead of a public key of the receiver of the certificate, the certificate is issued to an authority structure that contains also the public key of the used certificate revocation service.

When the certificate is used as proof of an authority, also another certificate

issued by the certificate revocation service must be added to the query. The certificate of the revocation service denotes that the public key which is authorized in the original certificate is non-revoked.

On the other hand, the administrator of the certificate revocation service must have some policy for the certificate revocation. For example, who can report that a key is unvalid and what kind of proof must be presented.

Evaluation

The administrator who issues the certificates must remember to add the public key of the certificate revocation service if the possibility to revoke certificates is required in the security policy of the service. That is, the revocation possibility must be organized beforehand. If this is forgotten in the creation of certificates, the certificates cannot be revoked.

The PolicyMaker certificates that allow revocation are more complex because the authority structure of the certificate must contain an explanation about the use of the certificate revocation service and this can only be done by writing it with the AWKWARD programming language. This makes the creation of certificates vulnerable to errors presented in section 6.1.1.

The administrator of the certificate revocation service must somehow be sure that a certificate needs to be revoked. Otherwise, an attacker can ask for revocation of valid certificates and block the usage of a service from authorized users.

KeyNote

How

KeyNote does not define how the certificates can be revoked but it can be used to create individual certificate revocation service.

Evaluation

The administrator must define the validity dates of a certificate very carefully because KeyNote does not provide any certificate revocation service.

The administrator of a resource must define how the KeyNote certificates are used when an individual certificate revocation service is created.

SDSI/SPKI

How

Usability problem	PolicyMaker	KeyNote	SPKI	TeSSA
Revocation must be taken care of beforehand	x	x	x	x
No revocation specified	(x)	x	-	-
Certificates that allow revocation require programming skills	x	-	-	-
Gaining assurance of the need of revocation	x	-	x	x

Table 6.5: Summary of Usability Problems of Case 5

The validity field of a SPKI certificate can give location and a public key of an online validation service. This information must be inserted to a certificate on its creation.

SPKI does not define how the certificate revocation service may be sure that a certificate really need to be revoked.

Evaluation

The administrator who creates the certificates must take care of certificate revocation beforehand by defining a location of certificate revocation service and a public key that must be used to sign the certificate revocation list.

TeSSA

How

TeSSA uses SPKI certificates and their certificate revocation system.

Evaluation

The problems are same as in the use of SPKI certificates.

Summary of Usability Problems of Case 5

Summary of the usability problems of the revocation in the different decentralized authorization systems is given in table 6.5. All systems requires that the revocation must be taken care when certificates are created because the certificates themselves gives the location of the certificate revocation service. The revocation itself is problematic, as is described in section 4.4.5.

6.1.6 Case 6: Checking Validity of a Set of Certificates

The checking of validity of the set of certificates that are presented as proof of compliance has two phases. Depending on the used decentralized authorization system, someone checks that each certificate is valid, and maybe someone else checks that the certificates form a valid proof of compliance.

PolicyMaker

How

In the PolicyMaker architecture, the application checks the validity of the certificates. Then the application sends the set of valid certificates to the compliance checker. The compliance checker reports if the certificate set gives the proof that the request complies with a given policy.

For checking validity of a single certificate, the application can asks from a certificate revocation service if the certificate is still valid. The revocation of PolicyMaker certificates is described above in the section 6.1.5. Other possibility to check the validity of certificates is to use "fresh" certificates, i.e. get a new certificate that denotes that the right is still valid.

The compliance checker does not understand the content of the certificates. The compliance checker will act as a blackboard where the proofs are added one by one, and maybe several times. Because the compliance checker does not understand the semantic used by the application, the administrator who writes the certificates must be carefully.

Evaluation

The administrator may relay that the compliance checker will also check the validity of each certificate, and leave this task undone. Forged certificates may then give rights to an entity who does not have proper authorization. The PolicyMaker compliance checker does not verify signatures and validity of the certificates, that task belongs to the the administrator of the application.

The administrator may relay that the compliance checker understand the semantics of the certificates. For this reason, she may have been careless when she issues the certificates. Then these carelessly created certificates ends up to give, for example, wider rights to a user.

KeyNote

How

In KeyNote, either the application or the compliance checker checks the validity of single certificates. If the compliance checker is used, it limits the possibility to choose the signature algorithm.

KeyNote compliance checker will check if a set of certificates gives proofs that the request comply with the policy of the application.

Basic KeyNote does not use certificate revocation, and thus checking that the certificate validity dates are correct is enough in addition to the checking of the signature.

The administrator must separately define the set of possibly results for the query evaluation. For example, the results can be "no_access", "limited_access", and "full_access". In addition to the defining these result values, the administrator must also put them in order from lesser rights to more rights.

Evaluation

The administrator must decide who checks the validity of the certificates. She must take care that both the validity of the individual certificate is checked and the authority that the set of certificates provide is checked.

If the administrator decides to send all certificates to the compliance checker of KeyNote, she must be sure that KeyNote knows the used cryptographic algorithms. Those certificates that are signed with unknown cryptographic algorithms are considered to be policy assertions, i.e. they are always true and part of the policy.

The administrator gives a set of possible compliance values. This set must be complete and give all possible values, otherwise the checking may fail: even when the set of certificates form a proof of compliance the result of the query is minimal trust. This set will be sent together with query and certificates to the KeyNote compliance checker. The result of the query is one of the values from this set.

When certificates are collected up, there might be certificates that does not proof authorization suitable for this query. These certificates does not need to be checked, because they do not influence to the query. Nevertheless, the administrator must be sure which certificates are essential for the query.

SDSI/SPKI

How

SPKI certificate chain validation is done by the protected resource. The protected resource checks that each certificate of the chain is valid, i.e. the signatures are valid, the validity time is not exceeded, and the certificate is not revoked. The chain is valid if the issuers and subjects of the certificates form a complete chain from the owner of the resource to the requester. The authorization that the chain of certificates issues is the intersection of all authorization fields of the certificates.

Evaluation

SPKI gives good instructions when a chain of certificates is valid and what kind of authorization the chain of certificates gives.

The validator must remember to check all certificates of the chain. It is not enough th check that the first certificate is issued by the source of authorization and the last certificate is issued to the requester. All certificates must be valid.

TeSSA

TeSSA uses SPKI certificates for trust management, and thus the usability problems are the same as in the case of SPKI.

Summary of Usability Problems of Case 6

Summary of the usability problems of the validity checking in the different decentralized authorization systems is given in table 6.6. The largest usability problem is poorly created authorization fields of certificates that may mislead the administrator to give an access to a resource to an unauthorized user.

6.1.7 Case 7: Privacy of Users

Usually, authorization systems do not use names to identify users of the systems. They authorize a public key, and the holder of the corresponding private key can proof that she has the rights. But, when a collection of certificates is gathered up for proving an authorization, the true identity of the keyholder may be revealed.

Usability problem	PolicyMaker	KeyNote	SPKI	TeSSA
Two place for checking the validity	x	(x)	-	-
Validity checker does not understand the content of certificates	x	x	-	-
Possible results defined in advance	-	x	-	-
Authorization is poorly defined in the certificates	x	x	x	x
All certificates needs to be checked	-	-	x	x

Table 6.6: Summary of Usability Problems of Case 6

PolicyMaker

How

PolicyMaker uses public keys to present users when an authorization is granted to them. But otherwise the use of identifiers is not limited. For example, the filter and authority structures can bind identifiers to the public keys. Thus, it is up to the application what kind of privacy protection is used in PolicyMaker.

When a user requests an action, the application collects certificates from various trusted third parties. Various unnecessary certificates may also end up to the application.

Evaluation

An administrator can issue certificate that reveals the identity (e.g. the name) of an entity for her own comfort because the names are easier to handle for humans. When an application collects information of the authorizations granted for the public key, it may also get the certificate binding the key to the name.

When an application seeks for the proof of compliance, it asks from TTPs certificates. Some TTP may give a certificate, that does not have anything to do with the requested action. The unnecessary certificates may reveal some additional information that may help the administrator to identify the user. This is not the purpose of an authorization system.

KeyNote

How

In Keynote, local constants can be used instead of complicated public keys to make certificate more readable for human users.

In KeyNote, the validator of individual certificate may be either the administrator or the compliance checker.

Evaluation

The administrator can use abstract names as nicknames for the public keys. For example, naming the entities by alphabetics does not reveal their identity but still help avoiding the use of incorrect keys. But if the administrator use real names as key nicknames, the true identity may be revealed.

If the compliance checker validates the certificates, it does not understand the content of the certificates, i.e. unnecessary information is unused. But if the administrator of an application verifies the certificates, she may understand also the authorization given in the unnecessary certitifates, and the identity of the keyholder may be revealed.

SDSI/SPKI

How

One objective of SPKI certificates is to protect the privacy.

Nevertheless, SPKI defines also name certificates. But the names of the certificates can be local names like "my mother" which do not actually reveal the identity of the entity.

Evaluation

Similarly to the KeyNote, the administrator must think about naming when the certificates are created. If she gives real names for a certificates, the identity may be revealed.

The requester will provide all necessary certificates in order to perform an action. Thus she herself is responsible if unnecessary certificates will be given to an administrator that checks the certificates.

TeSSA

How

Usability problem	PolicyMaker	KeyNote	SPKI	TeSSA
Privacy protection depends on application	x	x	x	x
Privacy protection depends on issued certificates	x	x	x	x
Name of an entity may be revealed accidentally	-	x	(x)	(x)
Additional certificates may reveal identity	x	x	-	x

Table 6.7: Summary of Usability Problems of Case 7

TeSSA uses SPKI certificates to express authorization. The certificates are stored in DNS service of both the issuer and the subject.

Evaluation

TeSSA protect privacy in two ways. It uses SPKI certificates to express authorization. Of course, the administrator who has issued the certificates can issue also name SPKI certificates.

The resource who wants to check that the user has access rights fetch the certificates from the DNS servers. It will got the first certificate from the user. Because the certificates form a chain that can also be followed from one DNS server to other, the resource will not get so many additional certificates. Still, it can get those certificates of the user that are stored in the same DNS servers than the needed certficates.

Summary of Usability Problems of Case 7

In principle, all the decentralized authorization systems protects the privacy of the authorized entity. The authorization is issued to a public key instead of a real life identifier of the entity. In practise, the privacy protection depends on the use of certificates.

Summary of the usability problems of privacy in the different decentralized authorization systems is given in table 6.7.

6.1.8 Case 8: Distinguishing Trusted Channels from Untrusted Channels

Decentralized systems consists of nodes connected through an unreliable network. Still, sometimes unsigned policies are used to denote authorizations. The administrators of the system must configure trusted channels between entities that use these unsigned policies.

PolicyMaker

How

PolicyMaker uses a compliance checker to check that a query complies with the security policy of an application. The compliance checker can be a part of the application but it can also be a separate entity in a different node. Because policy assertions of PolicyMaker are not signed, they need trusted channel when they are sent as part of a query from the application to the compliance checker that is located in the different node.

The administrator of the application must configure the system to send all unsigned certificates, i.e. policy assertions, through the trusted channel if the system uses external compliance checker.

Evaluation

The administrator must define what kind of protection is needed for the trusted channel. The definition can be derived from the upper layer security policy but the task is not trivial. PolicyMaker do not define how this can be done.

The setting up a trusted channel according to the requirements may also cause difficulties. Some external system must be used because PolicyMaker does not give support for this task.

KeyNote

How

Like PolicyMaker, KeyNote policy assertions are not signed. The entity that owns a resource, e.g. an application or a node, will deliver her own security policy to the compliance checker of KeyNote through a trusted channel.

The administrator of the system must create the trusted channel for the policy assertions. KeyNote does not define how the trusted channel is formed.

Usability problem	PolicyMaker	KeyNote	SPKI	TeSSA
Requires trusted channel because of integrity	x	x	-	-
Agreement about the used protection undefined	x	x	(x)	-
Need external systems for the trusted channel	x	x	x	-

Table 6.8: Summary of Usability Problems of Case 8

Evaluation

KeyNote must use external system for creating trusted channel between the compliance checker and the entity that owns the protected resource.

SDSI/SPKI

How

All the SPKI certificates are signed which guarantee the integrity of the certificates. If the confidentiality of the certificates is desired, a secure channel between entities must be created. However, the secure channel is different from the trusted channel required by PolicyMaker and KeyNote, because the integrity is already secured.

Evaluation

SPKI does not define how the external secure channel can be established.

TeSSA

How

The TeSSA architecture uses SPKI certificates which are always signed. In addition to the integrity, parties of the TeSSA architecture can use IPsec to secure the communication confidentiality.

Evaluation

Defining security association for IPsec is not a trivial task.

# usability problems	PolicyMaker	KeyNote	SPKI	TeSSA
Case 1	6	5	2	1
Case 2	5	5	3	3
Case 3	1	1	1	0
Case 4	3	1	2	2
Case 5	3	2	2	2
Case 6	3	3	2	2
Case 7	3	4	2	3
Case 8	3	3	1	0
Summary	27	24	15	13

Table 6.9: Number of found usability problems

Summary of Usability Problems of Case 8

Summary of the usability problems of the use of trusted channels in the different decentralized authorization systems is given in table 6.8.

6.2 Results of the Comparison

The decentralized trust management systems gives basically two different approaches to manage trust. PolicyMaker and KeyNote uses external compliance checker for validating given proofs, and SDSI/SPKI and the TeSSA architecture based on SPKI are authorization public key infrastructures.

Most important security tasks in a decentralized trust management are the issuing of certificates that denotes trust and checking that these certificates are valid and gives a requested right. Based on my analysis, the PKI-like approach is more usable than PolicyMaker approach for the administrators of the system. Table 6.9 presents how many usability problems was found in the analysis presented in this thesis.

In SDSI/SPKI, all trust is described similarly by signed certificates. The structure of the SPKI certificate clearly distinguishes the rights that can be delegated to others from the non-delegable rights. It is also easier to understand that the administrator can restrict the (delegable) right given to her by someone else than the concept where she can only give more rights than she has got.

Because the SDSI/SPKI certificates are always signed, they can be delivered from one entity to other entity of the decentralized system through untrusted

network. For example, a software agent does not need secure storage for her policy certificates because it can use external storage. If the policy certificates are unsigned, they must be stored in a secure system which itself needs carefully maintenance.

Even than an external common compliance checker may be multipurpose, it may cause more work to the administrator of the system. The creation of certificates that are evaluated without understanding the semantics of the certificates is hard. If the compliance checker does not check all the validity, i.e. the validity of single certificates and the validity of a set of certificates, the administrator can accidentally forgot to do her task, i.e. to validate the single certificates.

Nevertheless, the administrators and the developers need to understand well the security technology where the system is based on in order to create trustworthy system.

6.3 Experiences

The security functions of a decentralized systems are not the primary goals of the user of the system but secondary goals. There is no suitable method for easily to test the secondary usability functions such as usability of security [83].

In this thesis, I has used heuristic evaluation as basis of the comparison. The results are not entirely reliable and many usability problems may have stayed unrevealed. Heuristic evaluation will give good results if about five evaluators check the system.

Common Criteria suites well for identifying parts of a security system even though the CC is not used otherwise. No essential parts of the system can be forgotten, and thus all parts of little different systems are evaluated from the same starting point.

Chapter 7

Conclusions

Currently used centralized, identification based public key infrastructures has not conquered the world as has been expected. They does not scale well and they require unnecessary functions such as identification of users of the system. These systems can be replaced with decentralized trust management systems.

In decentralized trust management system, each entity can decide to who it trust and in what way. The entities can also states their trusts to others. The trust is not based on the identity of other party. In the end, the identity is unnecessary information. Only necessary information is that the entity who wants to perform an action has authorization to do so based on the given proofs.

The purpose of this thesis was to find out which of the decentralized trust management system supports best the users of the system by providing usable concepts.

Based on my analysis, SDSI/SPKI public key infrastructure with authorization certificates is the most usable system. It has common basis with the currently used identity certificate PKIs, and errors are not so likely to be done accidentally with SPKI.

However, my analysis has shown that the current decentralized authorization systems have many usability problems that must be solved before the systems are usable for a common user who does not understand well the underlying security technology. The concepts behind the system are complex which makes misunderstandings and errors likely to occur even among experienced users.

Bibliography

[1] Alfarez Abdul-Rahman and Stephen Hailes. A distributed trust model. In *Proceedings of 1997 New Security Praradigms Workshop*, pages 48 – 60. ACM Press, 1998.

[2] Anne Adams and Martina Sasse. Users are not the enemy. *Communications of the ACM*, 42(12):40–46, December 1999.

[3] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. *The AWK Programming Language*. Addison-Wesley, Reading, 1988.

[4] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers - Principles, Techniques, and Tools*. Addison-Wesely, 1986.

[5] Edward G. Amoroso. *Fundamentals fo Computer Security Technology*. Prentice Hall, 1994.

[6] Ken Arnold, Bryan O'Sullivan, Robert W. Scheifler, Jim Waldo, and Ann Wollrath. *The Jini Specification*. Addison-Wesley, June 1999.

[7] Tuomas Aura. Comparison of graph-search algorithms for authorization verification in delegation networks. In *Proc. 2nd Nordic Workshop on Secure Computer Systems NORDSEC'97*, Espoo, Finland, November 1997.

[8] Matt Blaze, Joan Feigenbaum, John Ioannidis, and A. Keromytis. The keynote trust-management system version 2. RFC 2704, IETF, September 1999.

[9] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, chapter The Role of Trust Management in Distributed System Security. Springer-Verlag, 1999.

[10] Matt Blaze, Joan Feigenbaum, and Angelis D. Keromytis. Keynote: Trust management for public-key infrastructures. In *Cambridge 1998 Security Protocols International Workshop, England 1998*, 1998.

[11] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *IEEE Conference on Security and Privacy, Oakland, CA, May 1996*, May 1996.

[12] Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance checking in the policymaker trust management system. In *2nd Financial Cryptography Conference, Anguila 1998*, pages 251–265. Springer-Verlag, 1998.

[13] Matt Blaze, John Ioannidis, and Angelos Keromytis. Compliance checking and ipsec policy management. Internet draft, IETF, March 2000. expired.

[14] Matt Blaze, John Ioannidis, and Angelos Keromytis. Trust management for ipsec. In *Proceedings of 2001 Symposium on Network and Distributed System Security (NDSS)*, 2001.

[15] Matt Blaze, John Ioannidis, and Angelos D. Keromytis. Trust management and network layer security protocols. In *1999 Cambridge Protocols Workshop*, April 1999.

[16] Matt Blaze, John Ioannidis, and Angelos D. Keromytis. Offline micropayments without trusted hardware. In *Financial Cryptography*, February 2001.

[17] Common Criteria Evaluation Board. Common criteria for information technology security evaluation, version 2.1, September 2000.

[18] Scott O. Bradner. The internet standards process – revision 3. RFC 2026, IETF, October 1996.

[19] Marc Branchaud. A survey of public-key infrastructures. Master's thesis, McGill University, Montreal, March 1997.

[20] Finnish Population Register Centre. Electronic identification. WWW-page, http://www.sahkoinenhenkilokortti.fi/, 2002. Referred 04-08-2002.

[21] Finnish Population Register Centre. Name service (nimipalvelu). WWW-page: http://192.49.222.187/nimipalvelu/defaul.asp, 2002. In Finnish, Referred 04-08-2002.

[22] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate chain discovery in spki/sdsi, December 2001. draft!!

[23] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, pages 644–654, November 1976.

[24] Donald Eastlake and Olafur Gudmundsson. Storing certificates in the domain name system (dns). RFC 2538, IETF, March 1999.

[25] Laura Lehtola (ed.). Tessa - telecommunications software security architecture. WWW page, URL:http://www.tml.hut.fi/Research/TeSSA/, 2000.

[26] Marcin Dobrucki (ed.). Internet protocol security at the telecommunications and multimedia laboratory. WWW page, URL:http://www.tml.hut.fi/Tutkimus/IPSEC/, 1997.

[27] Carl Ellison. Spki requirements. RFC 2692, IETF, September 1999.

[28] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylönen. Simple public key certificate. Internet-draft, expired, IETF, July 1999.

[29] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylönen. Spki certificate theory. RFC 2693, IETF, September 1999.

[30] Carl Ellison and Bruce Schneier. Ten risks of pki: What you're not being told about public key infrastructure. *Computer Security Journal*, XVI(1), 2000.

[31] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylönen. Spki examples. Internet-draft, expired, IETF, March 1998.

[32] Pasi Eronen. Security in the jini networking technology: a decentralized trust management appriach. Master's thesis, Helsinki University of Technology, March 2001.

[33] Pasi Eronen, Johannes Lehtinen, Jukka Zitting, and Pekka Nikander. Extending jini with decentralized trust management. In *OpenArch 2000*. IEEE Communications Society, 2000.

[34] Stephen Farrell and Russell Housley. An internet attribute certificate profile for authorization. RFC 3182, IETF, April 2002.

[35] Joan Feigenbaum. Overview of the at&t labs trust-management project. Technical Report 98.10.1, AT&T Labs - Research, 1998.

[36] Joan Feigenbaum. Towards an infrastructure for authorization, position paper. In *1998 USENIX Ecommerce Conference - Invited Talks Supplement*, pages 15–19, 1998.

[37] Joan Feigenbaum and Peter Lee. Trust management and proof-carrying code in secure mobile-code applications, a position paper. In *the DARPA Workshop on Foundations for Secure Mobile Code*, pages 48–55, March 1997.

[38] Matthew Fredette. *SDSI 2.0, The SDSI 2.0 Library and Tools*, 0.1 edition, February 1998.

[39] Thomas C. Greene. Ssl defeated in ie and konqueror. Web magazine: The Register, 12 August 2002. referred 12.8.2002.

[40] Object Management Group. The common object request broker: Architecture and specification. Technical report, Object Management Group, July 1998.

[41] JoAnn T. Hackos and Janice C. Redish. *User and Task Analysis for Interface Design*. John Wiley & Sons, 1998.

[42] Niklas Hallqvist and Angelos D. Keromytis. Implementing internet key exchange (ike). In *FREENIX Track 2000 USENIX Annual Technical Conference*, pages 201–214. USENIX, June 2000.

[43] Dan Harkins and Dave Carrel. The internet key exchange (ike). RFC 2409, IETF, November 1998.

[44] Tero Hasu. Storage and retrieval of spki certificates using the dns. Master's thesis, Helsinki University of Technology, April 1999.

[45] Austin Hill and Gus Hosein. The privacy risks of public key infrastructures. In *Data Protection Commissioner's conference in Hong Kong, September 13, 1999*, September 1999.

[46] Deborah Hix and Robert S. Schulman. Human-computer interface development tools: A methodology for their evaluation. *Communications of the ACM*, 34(3):74–87, March 1991.

[47] Telecommunication Standardization Sector International Telecommunication Union. X.509 series x: Data networks and open system communications, directory, information technology - open systems interconnection - the directory: Authentication framework. Technical report, International Telecommunication Union, Telecommunication Standardization Sector, August 1997.

[48] Hugh Johnson. *Suuri Viinikirja.* Tammi, 4 edition, 1995. English original: The World Atlas of Wine.

[49] Audun Jøsang. The right type of trust for distributed systems. In *New Security Paradigms '96 Workshop*, 1996.

[50] Kristiina Karvonen. Creating trust. In *The Fourth Nordic Workshop on Secure IT Systems (Nordsec'99), 1999, Kista, Sweden*, November 1999.

[51] Kristiina Karvonen. Enhancing trust online. In *PhDIT'99: Ethics in Information Technology Design. Second International Workshop on Philosophy of Design and Information Technology, 1999, Saint-Ferréol, Toulouse, France*, December 1999.

[52] Stephen Kent and Randall Atkinsin. Security architecture for the internet protocol. RFC 2401, IETF, November 1998.

[53] Stephen Kent and Randall Atkinson. Ip authentication header. RFC 2402, IETF, November 1998.

[54] Stephen Kent and Randall Atkinson. Ip encapsulating security payload (esp). RFC 2406, IETF, November 1998.

[55] Loren Kohnfelder. Towards a practical public key cryptosystem. Bachelor thesis, MIT, 1978.

[56] Tuomo Lampinen. Using spki certificates for authorization in corba based distributed object-oriented systems. In *The Fourth Nordic Workshop on Secure IT Systems (Nordsec'99), 1999, Kista, Sweden*, November 1999.

[57] Tuomo Lampinen. Using spki certificates for authorization in corba based distributed object-oriented systems. Master's thesis, Helsinki University of Technology, March 2000.

[58] Ilari Lehti and Pekka Nikander. Certifying trust. In *Theory and Practic in Public Key Cryptography (PKC'98)*, February 1998.

[59] Sanna Liimatainen and Teemupekka Virtanen. Distributed learning and management system for university courses. In *Proceedings of IFIP World Computer Congress 2002*, August 2002. Will be published, short paper.

[60] Douglas Maughan, Mark Schertler, Mark Schneider, and Jeff Turner. Internet security association and key management protocol (isakmp). RFC 2408, IETF, November 1998.

[61] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CPC Press, 1997.

[62] P. V. Mockapetris. Domain names – concepts and facilities. RFC 1034, IETF, Nobember 1987.

[63] Rolf Molch and Jakob Nielsen. Improving a human-computer dialogue. *Communications of the ACM*, 33(3):338 – 348, March 1990.

[64] Barry Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*. IEEE Computer Society Press, 1994.

[65] Jacob Nielsen. Finding usability problems through heuristic evaluation. In *Proceedings of ACM CHI'92 Conference*, May 1992.

[66] Jacob Nielsen. *Usability Engineering*. AP Professional, 1993.

[67] Pekka Nikander. *An Architecture for Authorization and Delegation in Distributed Object-Oriented Agent Systems*. PhD thesis, Helsinki University of Technology, 1999.

[68] Pekka Nikander and Jonna Partanen. Distributed policy management for java 1.2. In *Network and Distributed System Security Symposium, 1999, San Diego, USA*, February 1999.

[69] Pekka Nikander and Lea Viljanen. Storing and retrieving internet certificates. In Tønnes Brekne Svein J. Knapskog, editor, *NordSec'98, the Third Nordic Workshop on Secure IT Systems, 1998, Trondheim, Norway*, November 1998.

[70] Donald A. Norman. *The Invisible Computer, Why good products can fail, the personal computer is so complex, and information appliances are the solution*. The MIT Press, 1999.

[71] International Organisation of Standards. Ergonomic requirements for office work with visual display terminals (vdts) - part 11: Guidance on usability. ISO Standard 9241-11, ISO, 1997.

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

[73] Jonna Partanen and Pekka Nikander. Adding spki certificates to jdk 1.2. In Tønnes Brekne Svein J. Knapskog, editor, *NordSec'98, the Third Nordic Workshop on Secure IT Systems, 1998, Trondheim, Norway*, November 1998.

[74] Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland, and Tom Carey. *Human-Computer Interaction.* Addison-Wessley, 1994.

[75] Ronald Rivest. S-expressions. Internet draft, IETF, May 1997. Expired November 4, 1997.

[76] Ronald L. Rivest. Can we eliminate revocation lists? In *Proceedings of Financial Cryptography 1998*, 1998.

[77] Ronald L. Rivest and Butler Lampson. Sdsi - a simple distributed security infrastructure, April 1996.

[78] Bruce Schneier. *Applied Cryptography, Second edition.* John Wiley & Sons, Inc, 1996.

[79] Douglas R. Stinson. *Cryptography Theory and Practice.* CRC Press, 1995.

[80] Stephen Weeks. Understanding trust management systems. IEEE, 2001.

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

[82] Alma Whitten and J. D. Tygar. Usability of security: A case study. Technical report, Carnegie Mellon School of Computer Science, December 1998.

[83] Alma Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *8th USENIX Security Symposium*, August 1999.

[84] Phil Zimmermann. *The Official PGP Users Guide.* MIT Press, 1995.