HELSINKI UNIVERSITY OF TECHNOLOGY

Department of Computer Science

Ilari Lehti

SPKI-based Access Control Server

Master's thesis submitted for approval January 13, 1998 for the degree of Master of Science.

Supervisor Prof. Arto Karila

Instructor Lic.Tech. Pekka Nikander

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE
MASTER'S THESIS

Author and name of the thesis:

Ilari Lehti: SPKI-based Access Control Server

Date: 13.01.1998

Number of pages: 51

Faculty:
Department of Computer Science

Professorship:
Computer Networks (Tik-110)

Supervisor: Prof. Arto Karila

Instructor: Lic.Tech. Pekka Nikander

There are many approaches to ensure information security in a distributed environment. In addition to using strong cryptography and solid key management practices, we also need to be able to define trust relationships, policies and authorizations. A central security data storage and resource controller would be useful in reaching these goals. In this study, a prototype for such a data storage and resource controller is developed.

Simple Public Key Infrastructure (SPKI) is an IETF effort to create a worldwide authorization and delegation framework. While traditionally the task of certificates has been to certify identity, SPKI certificates certify authority. By binding cryptographic keys to the operations they are trusted to perform, we can avoid several problems present in identity certification and gain some new advantages, for example anonymity. SPKI principals are keys instead of names. Binding an access control decision to a name is rarely necessary.

It is the responsibility of the service user to arrange a sequence of certificates which shows that the user is authorized to use the service. The verifier processes the sequence using a mechanism called 5-tuple reduction. The result of a useful reduction is a certificate where the verifier itself is the issuer and the authorization is the intersection of the authorizations in the sequence. When certificates are used for access control, the user has to prove the control of the identifying key. This operation can be seen closing the certificate chain to a loop.

Today, the most widely used certification systems are PGP and X.509. The trust models of these two systems are very different. X.509 has a hierarchical certification tree in which everyone using the system must be able to trust. The PGP trust model is a decentralized "web of trust". The SPKI trust model is non-hierarchical and the type of trust can be defined.

A Security Policy Daemon (SPD) works as a policy information storage and provides functionality to securely control the usage of some service. It creates secure connections to other workstations and stores the connection information. The SPD may ask help for some access control decisions from a trusted policy server. It can also help IPSEC to provide network-layer security services.

The implementation language of the project was Java. It has many great features like platform independence and automatic memory management, but political issues may hinder it's development. Currently Java is not the language for performance critical applications. The SPD consists of a control part, security variables and different kinds of mechanisms. The control part waits for new tasks to appear in a task queue, which is filled by different adapters. Adapters are the interface mechanisms of the SPD that provide the connections to the outer world. The modularity of the chosen designs was enhanced using appropriate design patterns.

Keywords: SPKI, authorization, delegation, certificate, policy, SPD, Java, design patterns

Tekijä ja työn nimi:

Ilari Lehti: SPKI-pohjainen pääsynvalvontapalvelin

Päivämäärä: 13.01.1998 Sivumäärä: 51

Osasto: Tietotekniikan osasto

Professuuri: Tietokoneverkot (Tik-110)

Työn valvoja: Prof. Arto Karila

Työn ohjaaja: TkL Pekka Nikander

Hajautetun tietojärjestelmän turvallisuutta voidaan parantaa monilla eri tavoilla. Sen lisäksi, että käytetään vahvaa kryptografiaa ja hyviä avaintenhallintajärjestelmiä, täytyy myös olla mahdollisuus määritellä luottamussuhteita, turvakäytäntöjä ja valtuutuksia. Keskitetty turvatietojen tietovarasto ja resurssienhallinta olisivat tarpeen näiden tavoitteiden saavuttamiseksi. Tässä projektissa kehitetään prototyyppi tällaiselle tietovarastolle ja resurssienhallinnalle.

SPKI (Simple Public Key Infrastructure) on IETF:n yritys luoda maailmanlaajuinen valtuutus- ja delegaatiojärjestelmä. Kun sertifikaattien tehtävänä on perinteisesti ollut henkilöllisyyden varmistus, SPKI sertifikaatit varmistavat valtuuksia. Sidottaessa kryptografisia avaimia sallittuihin operaatioihin, voidaan välttää useita henkilöllisyyden varmistuksen ongelmista ja saavuttaa joitakin uusia etuja, esimerkiksi anonymiteetin säilyminen. SPKI yksilöt (principals) ovat avaimia eivätkä nimiä. Pääsynvalvontapäätöksen sitominen nimeen on harvoin tarpeellista.

On palvelun käyttäjän vastuulla koota sertifikaattijono, joka osoittaa hänet valtuutetuksi käyttämään kyseistä palvelua. Tarkistaja käsittelee jonoa käyttäen mekanismia nimeltä viisikkosupistus (5-tuple reduction). Hyödyllisen supistuksen tuloksena on sertifikaatti, jossa tarkistaja itse on julkaisijana ja valtuutus on jonon valtuutuksien leikkaus. Kun sertifikaatteja käytetään pääsynvalvontaan, käyttäjän on todistettava hallitsevansa identifioivaa avainta. Tämän tapahtuman voidaan ymmärtää sulkevan sertifikaattiketjun renkaaksi.

Nykyään yleisimmin käytetyt sertifiointijärjestelmät ovat PGP ja X.509. Näiden järjestelmien luottamusmallit ovat hyvin erilaiset. X.509:ssä on hierarkinen sertifiointipuu, johon kaikkien järjestelmän käyttäjien on luotettava. PGP:n luottamusmalli on hajautettu “luottamusverkko”. SPKI:n luottamusmalli on epähierarkinen ja luottamuksen laatu voidaan määritellä.

SPD (Security Policy Daemon) toimii turvakäytäntöjen tietovarastona ja tarjoaa mahdollisuuden turvallisesti hallita jonkin palvelun käyttöä. Se luo turvallisia yhteyksiä muihin työasemiin ja tallettaa yhteystiedot. SPD voi pyytää apua pääsynvalvontapäätöksiin luotetulta turvakäytäntöpalvelimelta. Se voi myös auttaa IPSEC:iä tarjoamaan verkkokerroksen turvapalveluja.

Projektin toteutuskielenä oli Java. Siinä on monia hienoja ominaisuuksia, kuten järjestelmäriippumattomuus ja automaattinen muistinhallinta, mutta poliittiset tekijät voivat haitata sen kehitystä. Tällä hetkellä Java ei sovellu suorituskyvyltään kriittisten sovellusten tekoon. SPD koostuu ohjausosasta, turvamuuttujista ja erilaisista mekanismeista. Ohjausosa odottaa uusien tehtävien ilmaantumista tehtäväjonoon, jota eri sovittimet (adapters) täyttävät. Sovittimet ovat SPD:n liitäntämekanismeja, jotka tarjoavat yhteydet ulkomaailmaan. Valittujen rakenteiden modulaarisuutta parannettiin käyttämällä sopivia suunnittelumalleja (design patterns).

Avainsanat: SPKI, valtuutus, delegointi, sertifikaatti, turvakäytäntö, SPD, design patterns

Acknowledgements

This master's thesis has been written while working as a research assistant in the Telecommunications Software and Multimedia Laboratory at Helsinki University of Technology. I especially want to thank my instructor Pekka Nikander, who knew how to encourage me in the moments when I was not sure whether the work was heading to the correct direction. I thank my supervisor Arto Karila for his comments and Arto, Pekka and Timo Aalto collectively for offering me the opportunity to research this interesting subject.

I want to thank my parents for always keeping the faith and trusting in me. I also want to thank my friends who helped me indirectly during this project.

Table of Contents

Glossary

ACL	Access Control List, a list that binds principals to access rights
API	Application Programming Interface
auto-certificate	Self-signed certificate that contains information about the principal
CA	Certification Authority. These are entities (e.g., businesses) that are trusted to sign (issue) certificates for other people (entities)
certificate	A signed statement about the properties of some entity
Conduits+	Framework for communication protocol software
CRL	Certificate Revocation List, a list that contains the identifiers of those certificates that appear valid, but have been revoked
cryptographic transformation	A cryptographic algorithm and a set of rules surrounding the use of the algorithm in a particular context
DES	Data Encryption Standard. A symmetric key encryption and decryption algorithm
digital signature	A digital signature is computed from some data and the private key of an entity
digitally signed	If some data is digitally signed it has been stored with the "identity" of an entity, and a signature that proves the entity knows about the data
DNS	Domain Name System. The on-line distributed database system used to map human-readable machine names into IP addresses and vice versa
entity	An entity is a person, organization, program, computer, business, bank or something else that has separate and distinct existence
entropy	the degree of disorder or uncertainty in a sys-

	tem
hash function	A function that maps a variable-length data block or message into a fixed-length value called a hash code
host security policy	Expression of a security policy in a host system defined by an organization. Host security policy defines the security requirements to be fulfilled. It determines what security services should be used when communicating outside the host. An individual application or user may, in turn, modify their security requirements within the limits set by the security policy
IAB	Internet Architecture Board
IETF	Internet Engineering Task Force. A group of people closely connected to the IAB who work on the design and engineering of TCP/IP and the global Internet
IP	Internet Protocol
IPv4	Internet Protocol version 4 (the currently used IP version)
IPv6	Internet Protocol version 6 (IP next generation)
IPSEC	Internet Protocol Security. A security extension of the Internet Protocol which is optional in IPv4 and mandatory in IPv6
ISAKMP	Internet Security Association and Key Management Protocol
ISO	International Organization for Standardization, formulates and publishes standards
MAC	Message Authentication Code. An authenticator that is a cryptographic function of both the data to be authenticated and the secret key. Also referred to as a cryptographic checksum
MD5	Message Digest algorithm 5. A one-way hash

	algorithm
Oakley	The Oakley Key Determination Protocol
PGP	Pretty Good Privacy, a public key based software package for data encryption. Used widely to secure electronic mail
PKI	Public Key Infrastructure. Needed to associate public keys to identities or authorizations
public key	Public keys are pieces of information associated with a particular entity, and are intended to be known to everyone who needs to have trusted interactions with that entity
public-key cryptography	aka asymmetric cryptography. A form of cryptosystem in which encryption and decryption are performed using two different keys, one of which is referred to as the public key and one of which is referred to as the private key
principal	An entity (e.g. person, workstation, process, device, etc.) in a distributed computer system
private key	Private keys are pieces of information that are supposed to be known only to a particular entity (i.e., kept secret). In a typical public-key cryptosystem, a private key corresponds to exactly one public key
RFC	Request For Comment. The name of a series of notes that contain surveys, measurements, ideas, techniques, and observations, as well as proposed and accepted TCP/IP protocol standards
RSA algorithm	A public-key encryption algorithm based on exponentiation in modular arithmetic
SA	Security Association
SDSI	Simple Distributed Security Infrastructure
security association	A Security Association (SA) is a relationship between two or more entities that describe how the entities will utilize security services to communicate securely. This relationship is represented by a the set of security information relating to a given network connection or

	a set of connections
security mechanism	A mechanism that is designed to detect, prevent, or recover from a security attack
SHA	Standard Hash Algorithm. A one-way hash function
signature	*see* digital signature
signed	*see* digitally signed
SPD	Security Policy Daemon
SPI	Security Parameter Index. An unstructured opaque index which is used in conjunction with the IP destination address to identify a particular security association
SPKI	Simple Public Key Infrastructure
SSL	Secure Sockets Layer. A security protocol that provides data encryption, server authentication, message integrity, and optional client authentication
symmetric cryptography	A form of cryptosystem in which encryption and decryption are performed using the same key (*cf* public-key cryptography)
TCP	Transmission Control Protocol
threshold subject	A certain number of subjects out of a larger set are needed to get the authorization
UDP	User Datagram Protocol
WWW	World Wide Web. A global hypertext based information system
X.500	An international directory service by ISO
X.509	An authentication framework by ISO

1 Introduction

"Hallo!" said Pooh, in case there was anything outside.
"Hallo!" said Whatever-it-was.
"Oh!" said Pooh. "Hallo!"
"Hallo!"
"Oh, there you are!" said Pooh. "Hallo!"
"Hallo!" said the Strange Animal, wondering how long this was going on.
Pooh was just going to say "Hallo!" for the fourth time when he thought that he wouldn't, so he said, "Who is it?" instead.
"Me," said a voice.
"Oh!" said Pooh.

In the above quote from [MIL28], we have an access control situation. Pooh, who is in control of the door, finds out that something wants to get in. An exchange of messages follows. The participants seem to lack a proper communication protocol and the messages remain pretty meaningless. At the end, an attempt of identification is made, but without proper credentials. Pooh, were he not a bear of no brain, should conclude that the voice has no authority to enter.

We are going to illuminate the ideas of authority delegation and certificate loops. We mainly focus on the IETF proposal called Simple Public Key Infrastructure (SPKI) [SPKI97] and on our implementation of an access control server based on that proposal. The implemented system allows trust and authority to be explicitly represented in the form of certificates.

Suppose Pooh would use such a system. After deciding on a door-opening policy, he could have issued credentials to trusted persons, perhaps even allowing them to further delegate this authority. Then the Strange Animal could have pushed a set of credentials under the door and Pooh, after checking their authenticity, could have let the stranger in. In the following subsections, we will explain these terms in the context of networked entities.

1.1 Security in a Distributed Environment

There finally is a lot of interest to the issues of network security. Several approaches have been made to improve the current security situation in the ever-growing Internet. There is IPSEC and SSL, there is PGP and X.509, there is this and there is that. Every proposal has its own point of view and its own solutions to the things that it happens to have chosen as *the problem* to be solved. The average user is probably very confused, just hoping that someone would do something that would make the situation better for him and his data. The sooner and easier the better.

The improvement may be within our reach quite soon, but unfortunately, it is probably not going to make things easy for the average user. Many, if not all, of the upcoming security solutions see public key cryptography as a very necessary part of achieving the goal. And with public key cryptography there come the difficult issues of key management and trust management. To keep one's keys intact, high-entropy passphrases are needed. High-entropy means difficult-to-guess, which in turn means difficult to remember. Trust management includes a lot of subtle issues that define who or what keys you can trust; and usually you can not. If software developers someday succeed to hide such details from the end-users without risking their data, that really is a good accomplishment. Until that day, a lot of new things need to be learned by a lot of people, just when computers were about to become the easy-to-learn toy for everyone. Luckily, it will be worth the effort.

What exactly is meant by network security? Let us consider a client-server situation where a user wants to use some service. We can set several requirements for a secure network service. [KAR91] lists the most important security requirements from the user's and from the service provider's point of view. [NIK] adds some more and groups them into different categories. Taking from both of these, we write a list of general requirements that applies to both the service user and the service provider. Depending on the service type, either entity may need to know one or more of the following:

- That the peer system is able to perform its task and to deliver the results in the intended way.
- That nobody else can listen to (and understand) the session.
- That nobody can undetected delete from, change, or add to the information transferred.
- That the identity of the other party is that claimed and that it remains the same throughout the session.
- That we can check whether the other party has the right to do something (access or provide the service, for example).
- That the events taken place during the session can, beyond reasonable doubt, afterwards be proved to an impartial judge.

These requirements are provided by the security functions *Availability*, *Confidentiality*, *Integrity*, *Identification*, *Authorization* and *Non-repudiation*, respectively.

While a lot can be done to improve the availability of a service, this function can not be fully assured in the kind of networks that we have today. Data integrity is important for most applications and often closely connected with identification. Confidentiality is also important, but not always necessary. However, there are applications where confidentiality is even more important than integrity. What we call identification is often called authentication, but we feel that authentication is a more general term; so, identification here means authenticated identity (cf [NIK]). Authorization determines the rights of the user and it may or may not be based on identification. Non-repudiation can be seen as a stronger level of trust compared to the situation where one can only assure oneself about something (which is called authentication in our terminology).

1.2 Threats and Attacks

Edward Amoroso does a creditable job when describing different security concepts in [AMO94]. Citing from this work we present the following definitions:

A **threat** to a computer system is any potential occurrence, malicious or otherwise, that can have an undesirable effect on the assets and resources associated with a computer system.

A **vulnerability** of a computer system is some unfortunate characteristic that makes it possible for a threat to potentially occur.

An **attack** on a computer system is some action taken by a malicious intruder that involves the exploitation of certain vulnerabilities on the part of an attacker.

Threats can be categorized similarly to the security requirements mentioned before. For example, a threat that causes the integrity of some data to potentially be in danger is called an integrity threat, and a threat towards the availability of a system is called a denial of service threat.

Attacks towards communications can be divided into passive and active attacks. This division is based on whether the attack involves modification of data. Masquerade and replay are examples of active attacks. Masquerading means deceiving another participant to

believe that he or she is communicating with a different person.
Eavesdropping and traffic analysis are examples of passive attacks. [STA95]

1.3 Trust Models

The security requirements can be fulfilled by using suitable algorithms, protocols and operating procedures. Would the goal of security be completely reached then? No, we still need to choose who to trust. Even if we have a confidential channel, it is not much of help if the other end of the channel decides to publish our secrets.

Trust is a belief that an entity behaves in a certain way. Trust to a piece of machinery is usually a belief that it works as specified. Trust to a person means that even if someone has the possibility to harm us, we believe he/she chooses not to. The trust requirements of a system form the system's trust model. All computer systems, protocols and security frameworks have trust requirements, i.e. they have trust relationships that the user needs to share. We may need to have some kind of trust to the implementor of a software which source code is not public, trust to the person with whom we communicate on a network, trust to the computer hardware that it provides us with correct computation results, and so on. The trust relationships that we are interested in here are those between us and some other networked entities. Those other entities may be fellow human beings or machines providing some service.

It is of equal importance to analyze the trust requirements of a protocol or a framework as it is to analyze the soundness of the technical methods that it uses to achieve security. Trust requirements may be analyzed in several ways. We may consider one generic notion of trust and find the entities that we need to trust. The next alternative would be to classify different types of trust and define the ways we need to trust each entity [YKB93]. Yet another refinement might be to define a degree of trust needed towards the other entities [BBK94]. The ways to categorize and analyze trust may be called trust modeling, but with a trust model we mean the set of trust relationships of a system.

As an example for analyzing trust, a bank may tell me that the most appropriate way to protect the data traffic between my workstation and their server is to use a cryptosystem where they provide me a good-quality keypair. (This is among the most reasonable security-related offers that banks seem to provide.) Not only need I trust this bank's capability to make good keys, I also need to trust the bank completely, because they have the key that is supposed to be secret and identifies the service user as me. Many people are willing to accept that, it is a bank after all. But the truth is that this trust extends to every employee of the bank that has access to those keys. Why would I want to take such a risk, if it is possible for me to create my own key, not known to anyone else? Of course, creating my own keys requires me to have enough trust in my own key generation software and hardware.

1.4 Security Policies

Closely related to the concept of trust is the concept of policy. A security policy is a manifestation of laws, rules and practices that regulate how sensitive information and other resources are managed, protected and distributed. Every entity may be seen to function under its own policy rules. In many cases today, these rules are very informal, probably even unwritten. The policy of an entity or part of it is often derived according to some hierarchy, e.g. next level in a corporate hierarchy.

Security policies can be meaningful not only as an internal code of function, but as a published document which defines some security-related practices. This could be important information when some outsider is trying to decide whether an organization can be trusted in some respect. This is one situation where it is useful to define the organizational policies strictly and systematically, e.g. to have a formal policy model.

Another and a more important reason to have a formally specified policy is that a lot of the policy information should be directly accessible by the workstations and their software. Having a policy control enforced in software rather than relying on the users to follow some memorized rules is essential if the policy is meant to be followed. A lot of policy rules are already present in the operating systems, protocols, applications and their configuration files. A central policy storage and a policy supervising software would make these and other policy settings easier to maintain and analyze. We will call such a policy supervising software a Security Policy Daemon.

1.5 Digital Certificates

A certificate is a signed statement about the properties of some entity. In a digital certificate the signature is a piece of data computed from the certificate data and the key of the issuer. If a certificate states something about the issuer, it is called a self-signed certificate or an auto-certificate.

Traditionally, the job of a certificate has been to bind a public key to the name of the keyholder. This is called an identity certificate. It typically has at least the following fields: the name of the issuer, the name of the subject, the associated key, the expiration date, a serial number and a signature that authenticates the other fields of the certificate.

Not all applications benefit much from a name binding. Therefore certificates can also make a more specific statement, for example, that some entity is authorized to get a certain service. This would be called an authorization certificate. In addition to the fields in an identity certificate, more detailed validity fields are often needed. The "associated key" -field is replaced by the authorization definition field. The issuer and the subject are typically not defined with names but with keys.

In all certificate systems, but especially in identity certificates, it is important to choose a proper name space. The naming should be unique in the sense that no two principals have the same name, though one principal may have several names. Names should be permanent, if the principal so decides, so no enforced name changes should occur. In the authorization certificate naming schemes, the name binding may be early or late binding. Late means that when authority is bound to some name, the name need not yet be bound to a key, but this can be done afterwards.

Certificates and trust relationships are very closely connected. The meaning of a certificate is to make a reliable statement concerning some trust relationship. Certificates often form chains where the trust propagates transitively from an entity to another. The verifier needs to be able to trust to the first certificate of the chain for the chain to be useful. In some systems, this trust is implicit. Some systems make it explicit by requiring the first certificate of the chain to be issued by the verifier.

In the case of an identity certificate, the certificates express trust to some name binding. Authorization certificates often delegate some property or right of the issuer to the subject. It is also desirable that the system allows to limit the delegated authority in the middle of a path.

It would be useful to have some kind of global Public Key Infrastructure (PKI) for the storage and management of public keys and certificates. There are several proposals for such systems, but this area has been found to be very problematic. We will later study some of the proposals in more detail.

1.6 Outline of This Work

The rest of this work is organized as follows. In the next chapter, we will take a close look to the SPKI and some of the new ideas behind it. We will also present an example of certificate loops. In chapter 3 we are going to discuss the currently most popular certificate systems and compare the different certification approaches. Chapter 4 brings us an overall view of the Security Policy Daemon, which is the subject of the implementation part of this project. Chapter 5 discusses the actual implementation and the motivations behind the implementation choices. The last chapter sums up the key ideas the and the most important experiences gained during the work.

2 Simple Public Key Certificate

2.1 Status

The larger interest for an authorization certificate infrastructure started in 1996, when three different proposals were published. They were not totally isolated efforts, but instead took influences from and gave credit to each other. These were the SDSI [SDSI96] by Rivest and Lampson, the PolicyMaker [BLA96] by Blaze, Feigenbaum and Lacy, and the SPKI by the new IETF Working Group of the same name.

The task of the SPKI working group is to develop standards for a public key certificate format, associated signature and other formats, and key acquisition protocols. The formats and associated protocols are to be simple to understand, implement, and use. The working group will publish its first RFC soon. The current draft, named "Simple Public Key Certificate", is concerned only with certificate, hash, key and signature formats, using the certificates both for verification of identity and for proof of authorization. It does not cover the other elements of a full public key infrastructure: key/certificate storage and acquisition. [SPKI97]

The SPKI is intended to provide mechanisms to support security in a wide range of Internet applications, including Internet security protocols, encrypted electronic mail and WWW documents, payment protocols, and any other application which will require the use of public key certificates and the ability to access them. It is intended that the Simple Public Key Infrastructure will support a range of trust models. [SPKI97]

2.2 Principals and Naming

The SPKI principals are keys. Delegations are made to a key, not to the keyholder. By definition, the keyholder of a key has sole possession of the private key. Therefore, either this private key or the corresponding public key can be used as an identifier (a name) of the keyholder. For any public key cryptosystem to work, it is essential that a principal will keep its private key to itself. If the principal does not keep the private key protected (if the private key gets out to others to use) then it is not possible to know what entity is using that key and no certificates will be able to restore the resulting broken security.

However, long keys are inconvenient for a mere human to handle. Using names instead of keys is necessary at least in the user interfaces. Names in the certificates also allow late binding, which means that one can attach some properties to a name and later define or change the name-key binding. Names can also serve the purpose of a certain role. Therefore it is useful to also have other names than keys. SDSI abandoned the idea of a global name space and introduced linked local name spaces. SPKI uses this same naming scheme, where everyone can attach names to keys and every name is relative to some principal. The names can be chained so that speaking about Alice's Mother consists of a name Alice in my name space and a name Mother in Alice's name space.

Names need not always refer to single users but they can be used to refer to sets of users as well. If an issuer makes a name-key binding while another similar binding to the same name is still valid, these two certificates do not conflict but define a group with at least two members. It is, of course, possible to revoke the earlier binding, if the intention is to change the binding to a new one. In fact, because SPKI certificates always increase the subject's properties, we will never have to deal with a situation where two certificates would conflict.

An SPKI certificate is closer to a "capability" as defined by [LANDAU] than to an identity certificate. There is the difference that in a traditional capability system the capability itself is a secret ticket, the possession of which grants some authority. An SPKI certificate identifies the specific key to which it grants authority. Therefore the mere ability to read (or copy) the certificate grants no authority. The certificate itself does not need to be as tightly controlled. [SPKI97]

From the certificate usage point of view, the involved principals are called the prover and the verifier. It is the responsibility of the prover to present the needed certificates. Based on these, the verifier determines whether access is granted.

2.3 Certificate Format

SDSI used S-expressions, a recursive syntax for representing octet-strings and lists. An S-expression can be either an octet-string or a parenthesized list of zero or more simpler S-expressions. The current SPKI proposal uses the same construct; more specifically its canonical form. No empty lists are allowed, and each list must have a byte-string as its first element. That first object is the type of the object represented by the list. Here is an example of a canonical S-expression:

(4:test10:abcdefghij5:123455::: ::)

This S-expression consists of four byte-strings: "test", "abcdefghij", "12345" and ":: ::". Each string is preceded by its length.

The current SPKI format is still a draft and it is possible that it goes through minor changes. We will next sketch the current syntax and present some of the rationale behind it.

2.3.1 Sequence-construct

The core of the syntax is called a sequence. It is an ordered collection of certificates, signatures, public keys and opcodes taken together by the prover. A signature refers to the immediately preceding non-signature object. A certificate may be followed by multiple signatures. Opcodes are operating instructions, or hints, to the sequence verifier. They may, for example, say that the previous item is to be hashed and saved because there is known to be a hash-reference to it in some subsequent object.

2.3.2 Certificate definition subsets

The SPKI draft specifies different degrees of complexity for the certificates. The subset definitions are meant primarily for closed environments. In addition to the Bare-bones SPKI definition there are three extensions independent of one another:

- Simple SPKI/SDSI (addition of SDSI names)
- Tag algebra (addition of the *-form construct)
- Threshold subjects (addition of multiple-subject forms)

Everyone is strongly encouraged to use the full definition, which includes all of these extensions. We omit the detailed descriptions of the subsets and concentrate on the full definition.

2.3.3 Fields

The fields of an SPKI certificate are: *version*, *cert-display*, *issuer*, *issuer location*, *subject*, *subject location*, *delegation*, *tag*, *validity*, and *comment*. All of these, except issuer, subject and tag, are optional fields.

Version is the version number of the format. Cert-display is a display hint for the entire certificate. Issuer is a normal SPKI principal, i.e. a key or a hash of key. The location-fields define a place where to find additional information about that principal. For example, the issuer location may help the prover to track down previous certificates in the chain. Delegation is a true/false -type field defining whether the authority can be delegated further. Comment-field allows the issuer to attach human readable comments. Validity defines the conditions which must be fulfilled for the certificate to be valid. It is possible to define a time range of the validity and a detailed description of the chosen validation method. The possible methods are CRL, revalidation and one-time validation. Certificate Revocation List (CRL) is a list of invalid certificates, the revalidate form validates the certificate for a certain time period and the one-time form is a validation only for this specific occasion.

The most complex fields are the subject and the tag. The subject can be either a key, a hash of key, a keyholder, an SDSI name, an object or a threshold subject. A keyholder subject refers to the flesh and blood (or iron and silicon) holder of the referenced key instead of to the principal (the key). A threshold subject defines N subjects, K of which are needed to get the authority.

The tag contains the exact definition of the delegated authority. SPKI defines a tag algebra for expressing sets of permissions so that a standard, simple algorithm in the verifier can intersect tag-fields.

2.4 5-tuple Reduction

Five of the certificate fields have relevance for security enforcement purposes: issuer, subject, delegation, authority (tag) and validity. These security-relevant fields can be represented by a "5-tuple":

(I,S,D,A,V)

In the basic case, a pair of 5-tuples can be reduced as follows [SPKI97]:

(I1,S1,D1,A1,V1) + (I2,S2,D2,A2,V2)

becoming

(I1,S2,D2,A,V)

if S1=I2 (meaning that they are the same public key)
and (D1 = TRUE)
and A = intersection(A1,A2)
and V = intersection(V1,V2)

The validity intersections are trivial. The authority intersections are defined by the tag algebra. S-expressions of the form (* ...) are called *-forms. An S-expression that does not contain any *-forms is *-free. Every S-expression represents a set of *-free S-expressions. Examples of *-forms:

(*) is the set of all S-expressions
(* null) is the empty set
(* set ...) is the union of the meanings of the list elements
(* intersect ...) is the intersection of the meanings of the list elements

Sets of S-expressions can also be built by reordering, adding or deleting parts:

(* append s) is all the lists that start with list s

(* reorder s) is the lists obtained by reordering the list s

(* reorder -insert s) and (* reorder -delete s) are the lists obtained by reordering s and possibly inserting or deleting some elements

When composing certificate chains, the tag fields are intersected by a standard intersection algorithm. The user does not have to specify an intersection algorithm for his tags, but one does have to write the used tags in such a way that the standard intersection algorithm gives the desired behavior [SPKI97].

By reduction, some chains of authorization statements will be reduced to the form:

(Self, X, D, A, V)

where "Self" represents the entity doing the verification. Any authorization chain not reducing to a 5-tuple with Self as an issuer is not relevant to decisions by Self.

2.5 Typical Transaction

So far we have talked about certificates and certificate chains. When a service is used, it must be known that the contacting user really is the entity that is authorized by the given chain. At this point, if not before, the user proves the possession of the identifying key to the verifier. This action closes the authorization chain so that every useful chain can be seen as a loop.

Figure 1 shows a basic authorization loop implemented with SPKI certificates. The three certificates are possibly created long before they are used. The server has delegated the permission to access the service to its policy administrator. In this certificate, the delegate-property is set to true so that the policy admin may make further del-

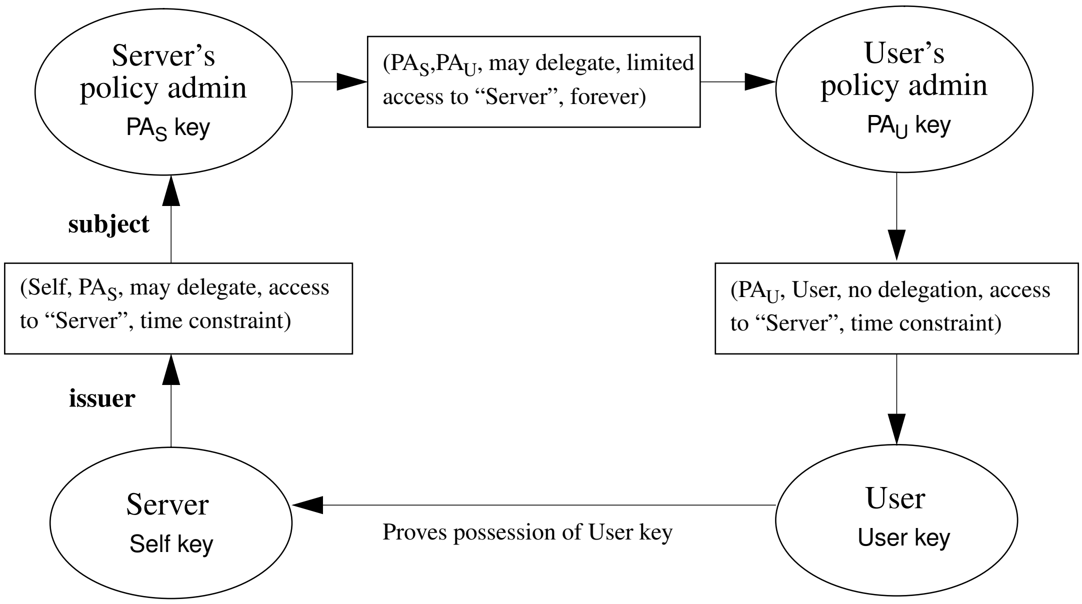

FIGURE 1. Basic authorization certificate loop

egations. The permission propagates through the policy administrators and their certificates to the service user.

To be complete, this example needs also another loop. Figure 2 shows the corresponding service identification loop. This is used by the user to authenticate the identity of the service provider, and it may even be completed before the service is used for the first time. This loop does not have to travel the same path as the authorization loop. The middle nodes need necessarily not be official Certification Authorities (CAs) as perhaps suggested by the graph, but the assurance of the server identity and services may sometimes be gained via less official paths.

When the actual service usage takes place, the user provides the authorization certificates to the server. Some systems have proposed a central repository from where the server makes a search. SPKI could use such a storage. However, in the case of authorization certifi-

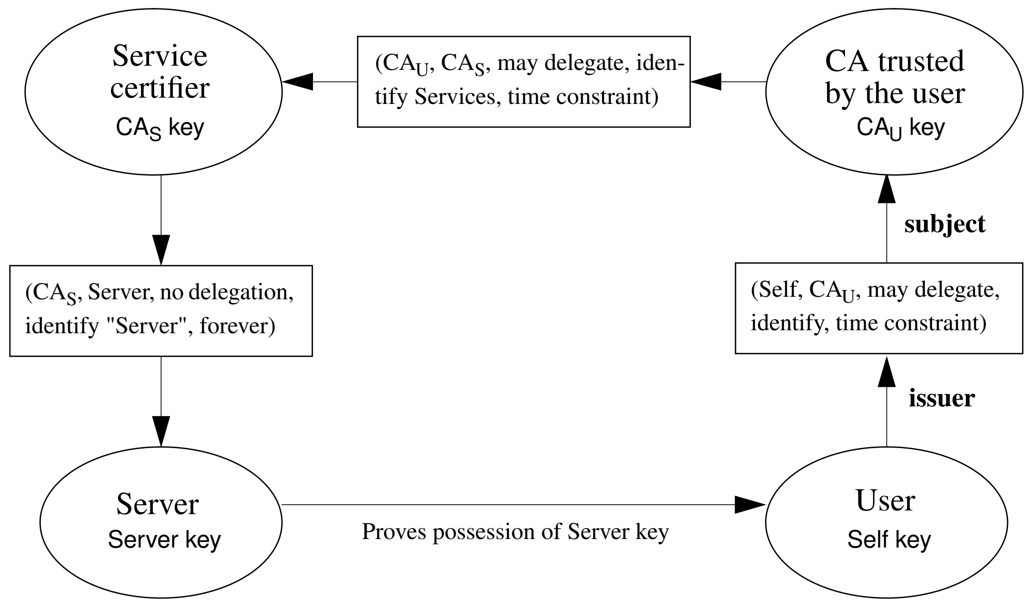

FIGURE 2. Basic service identification loop

cates , the central repository can be seen as a privacy threat unless it is somehow protected against general searches .

In addition to the server and the user, there may be other participants in the certificate usage process. Some other entity may reduce part of the chain and issue a certificate result certificate, which the server may use as part of the final reduction. One reason for using result certificates may be that the full chain contains certificate formats which the server does not understand. The server may also make its own result certificates for performance reasons, so that it need not make the same reduction several times.

3 Certificate Analysis

In this chapter we describe some concrete certificate infrastructure proposals and compare them. As already told, these systems can be divided into two main categories: those certifying identity and those certifying a specific authorization. In addition to that, the systems have important distinctions in their initial trust requirements and trust hierarchies.

3.1 Certifying Identity

We did earlier mention the name binding function of an identity certificate. More generally, with a binding we mean some important relationship or connection between two or more aspects of a system. In the case of certification, such aspects include the person, the person's name, the cryptographic key, or the remote operation about to be performed. We would wish all the relevant connections to be strong bindings instead of weak ones.

Figure 3 shows the bindings of an identity certification system used for access control. The certificate chain does the binding of a key to a name. Name is authorized to perform some operation according to an access control list (ACL), stored in the service provider's private storage. When the service is used, a key challenge between the participants binds a key to the operation to be performed. These three bindings can be made strong with appropriate cryptographic mechanisms.

The strength of a binding is here seen to come from the cryptographic strength of the used cryptosystem. In reality the strength also depends on the trust relationships involved and is always subjective. We could draw the private key and the public key separately. This

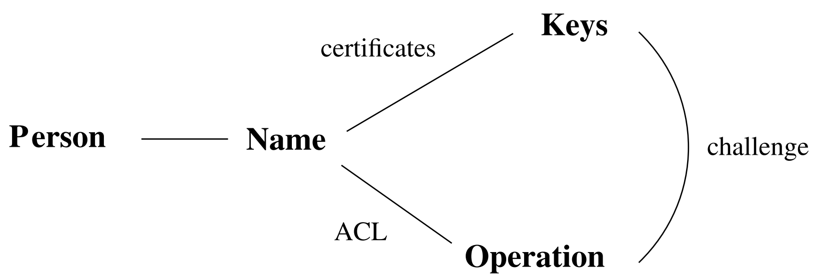

FIGURE 3. Identification certificate bindings

would bring one more step to the 'loop'. The binding between these two keys is the strongest binding of all.

But, the person-name binding is subject to some vulnerability. It is rarely taken into serious consideration at all, but usually assumed to be common knowledge. The larger the name space, the less likely that anyone would know the name of a specific individual. Alice might suspect that her old friend Bob Smith has a name in the global directory service, but there are so many people named Bob Smith in the world that it is unlikely Alice would know which of the thousands of Bob Smiths was in fact her old friend [SPKI97]. In a few special cases where you know what to look for, this binding can intuitively be considered strong, but it's never strong in the sense of cryptographic strength.

It can be argued that identity-based certificates create an artificial layer of indirection between the information that is certified (which answers the question "who is the holder of this public key?") and the question that a secure application must answer ("can we trust this public key for this purpose?") [BLA96]. Currently each application has to re-implement the mapping of names to actions that they are trusted to perform.

Of the existing identity certifying systems around, two have been taken into quite a common usage. PGP [PGP] has been a success since its introduction. X.509 [X509], while not taking an actual flying start, has lately gained some popularity despite the lack of any working global scale X.500 directory service.

3.1.1 PGP

In the PGP system, a user generates a key pair that is associated with his or her unique ID. Keys are stored in key records. A key record contains an ID, a public or a private key, and a timestamp of when the key pair was created. Public keys are stored on public key rings and private keys on secret key rings. Each user must store and manage a pair of key rings. [BLA96]

If user A has a copy of user B's public key record that she is confident of having not been tampered with since B generated it, A can sign this copy and send it back to B, who can give this 'certified' key-record to other users, such as user C. A thus acts as an introducer of B to C. Each user must tell the PGP system which individuals are trusted as introducers. Moreover, a user may specify the degree of trust in each introducer. [BLA96]

3.1.2 X.509

As in PGP, X.509 certificates are signed records that associate users' IDs with their cryptographic keys. Even if they also contain the names of the signature schemes used to create them and the time interval in which they are valid, their basic purpose is still the binding of users to keys.

However, X.509 differs sharply from PGP in its level of centralization of authority. While anyone may sign public-key records and act as an introducer in PGP, the X.509 framework postulates that everyone will obtain certificates from an official CA. When user A creates a key pair, she has it and the rest of the information certified by one of more CAs and registers the resulting certificates with an official directory service. When A later wants to communicate securely with B, the directory service must create a certification path from A to B. The X.509 framework rests on the assumption that CAs are organized into a global "certifying authority tree" and that all users within a "community of interest" have keys that have been signed by CAs with a common ancestor in this global tree. [BLA96]

The latest version, called X.509 v3, has a mechanism called certificate extensions. Using these extensions it is technically possible, even if not convenient, to use X.509 certificates for authorization purposes. Neither the specifications [IPKI97] nor the current usage of the system gives any support for such a practice, though.

3.1.3 Name spaces

As identity certification means binding a name to a key, the first concern is the choice of a name space. PGP really has no name space, which means that the name space is flat and any names can be used. It is common practice to use a name of the form (Full Name, EmailAddress).

X.509 uses hierarchical naming based on the X.500 directory service, which is considered to be realized so that independent organizations and their subdepartments would take care of naming their employees uniquely. X.500 has not come to pass and given the speed with which the Internet adopts or rejects ideas, it is likely that X.500 will never be adopted [ELL96b]. Organization-based naming scheme also has the undesirable property that if one changes a job, one's name will change at the same occasion.

One of the main obstacles for a wide acceptance of a global distributed directory service like X.509 is that most companies do not want to reveal the details of their internal organization, personnel etc. to their competitors. This would be the same as making the company's internal telephone directory public and, furthermore, distributing it in an electronic form ready for duplication and automatic processing [KAR91]. A controlled and secure directory service would be possible to create, but apparently there is currently no large market demand for it.

3.1.4 Trust models

"As PGP has grown in popularity, a decentralized 'web of trust' has emerged. Each individual is responsible for acquiring the public-key certificates he needs and for assigning degrees of trust to the introducers he gets them from. Similarly, each individual must create his own key pair and disseminate his own public key. This 'grass roots' approach rejects the use of official certifying authorities that sign public keys of individuals (and those of other certifying authorities) and thereby act as 'trust servers' for the users of those keys." [BLA96]

The X.509 trust hierarchy is shown in Figure 4. One of its problems is that it has a centralized trusted entity as a root that everyone with their differing needs should choose to trust. The system also implies everyone's trust to all the nodes of the certifying tree. In X.509, the authorization decisions are separate from the certification of identity even though all CAs must be trusted with respect to all authorizations.

The strong part of X.509 is that the protocol for finding someone's public key is well defined, as long as the assumed hierarchy exists.

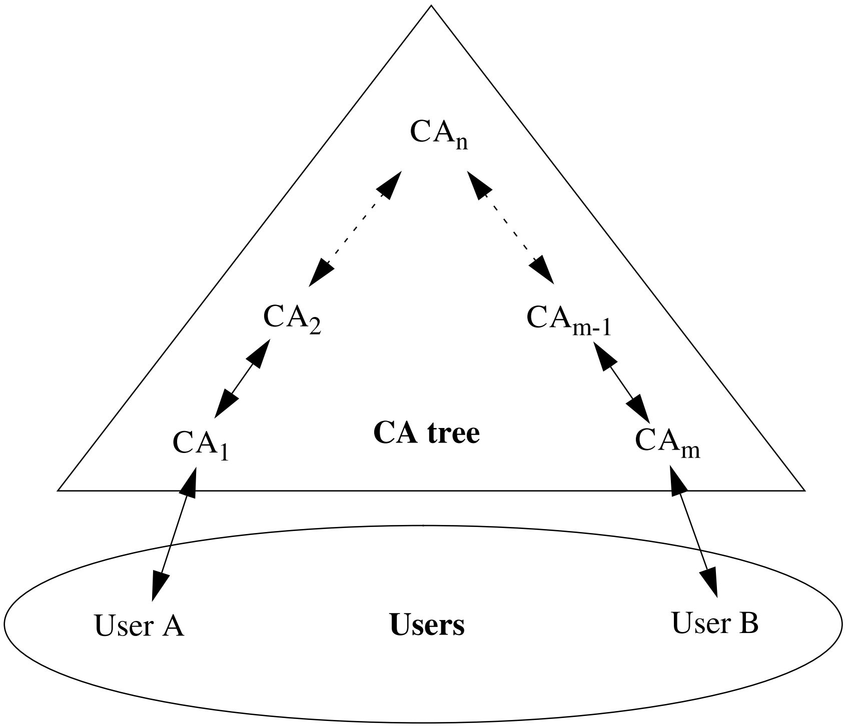

FIGURE 4. X.509 trust hierarchy

The non-hierarchical approaches leave something to be desired in this respect.

An interesting effort to combine the hierarchical trust model and the web of trust was made in the ICE-TEL project [CHA97]. The ICE-TEL calls its trust model a *web of hierarchies*-model. It is based on security domains, which can be as small as single-user domains. A security domain encapsulates a collection of objects that all abide by the rules defined in the domain's security policy. Then each domain can freely choose what other domains to trust. This has the advantage that there is no single trusted root of the hierarchy. ICE-TEL claims to handle authorizations as well, but is essentially an identity-based system that relies quite strongly on the use of Certification Authorities.

3.2 Certifying Authorization

The bindings of an authorization certificate system are shown in Figure 5. The certificate chain binds a key to an operation. A key challenge also operates between an operation and a key, thus closing the certification loop. These two bindings are based on cryptography and can be made strong. Again, we have chosen to draw the two keys together for clarity. Drawing them separately would just lengthen the certificate loop by one step.

The person-key binding is different from the person-name binding in the case of identity certification. As discussed in the beginning of Section 2.2 on page 11, the person is the only one having access to the private key and the key has enough entropy so that nobody has the same key. Common names are not automatically unique even if we add company information or other such constructs. So, the identifying key is bound tightly to the person that controls it and all bindings are strong.

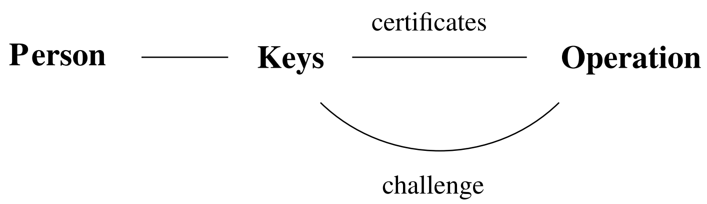

FIGURE 5. Authorization certificate bindings

The problem with the person-key binding is that from the service provider's point of view it looks like an undefined binding. The provider does not know who has control over the key. However, neither is this question essential in the most usual applications, nor can the traditional identity certification always answer this question. For the cases where the real physical identity of a keyholder needs to be known, [ELL96a] discusses different possibilities how to bind persons to keys without certification authorities. [SPKI97] proposes so-called process server certificates, issued by commercial enterprises, to aid in handling the extreme cases.

The feature of not having to bind keys to names is especially convenient in systems that include anonymity as a security requirement [BLA96, ELL96b]. It is easy for a user to create new keys for such applications, while creating an authorized false identity is (hopefully) not possible.

The most important authorization certification proposals are SDSI, SPKI and PolicyMaker. In the previous chapter we had a detailed discussion about SPKI. SDSI need not be described here, because although started separately, SDSI and SPKI have merged in their latest versions.

3.2.1 PolicyMaker

The goal of the PolicyMaker effort has been to separate the generic mechanism (provided by PolicyMaker) from application-specific policy (defined by each application). The PolicyMaker service appears to applications very much like a database query engine. It accepts as input a set of local policy statements, a collection of credentials, and a string describing a proposed trusted action. Depending on the credentials and form of the query, it can return either a simple yes/no answer or define additional credentials that would make the proposed action acceptable. [BLA96]

The basic function of a PolicyMaker system is to process queries of the form:

key_1, key_2, ..., key_n REQUESTS ActionString

A query is a request to determine whether a particular public key (or a sequence of public keys) is permitted to perform a particular action according to local policy. Action strings are application-specific messages that describe a trusted action requested by a (sequence of) key(s). PolicyMaker processes queries based on trust information contained in assertions. Assertions delegate authority on keys, working as certificates or policies. Assertions are of the form:

Source ASSERTS AuthorityStruct WHERE Filter

Here, Source indicates the authority, which can be either local policy or a public key of a third party. AuthorityStruct specifies the public key or keys to whom the assertion applies. Filter is the predicate that action strings must satisfy for the assertion to hold. [BLA96]

3.2.2 Trust models

These authorization-based systems are very general and flexible. They have no pre-defined trust hierarchy, but any user can define who to trust, which will lead to a PGP-like web of trust. In authorization certificates, the type of trust is always defined as well. The non-hierarchical approach contains fewer initial trust requirements to start with. It allows for single organizations to build their own web of policies and certificate servers inside the organization. After this it is possible to extend the trust to related organizations that you do your business with. It is immediately possible to start writing certificates for others to have when using your resources or to request certificates which allow the use of other services. Furthermore, you do not have to trust a particular CA if you do not want to, and you can choose to trust only some properties of a certain CA. Having the possibility to choose who to trust and in what respects allows for a great flexibility.

It is also important that delegated authority can be limited in the middle of an authorization path.

PolicyMaker is even more flexible than SPKI/SDSI. This can be seen either as a positive or a negative thing. All the mechanisms and conventions that are present in an SPKI-like system must be separately constructed in PolicyMaker filters every time they are needed. Filter complexity may make the system vulnerable to denial of service attacks.

A recommendation for certificates to bind keys to a certain task instead of certificates binding keys to a person can be seen in 'the explicitness principle', stated in [AN95], for example. It says that in order to make cryptography robust, everything (assumptions, goals, messages, etc.) should be stated as specifically as possible. It is more specific to define an operation for a key than to bind the key to a person.

4 Security Policy Daemon

As already briefly mentioned, we are in need of a central security data storage and a controller of the provided resources. Security Policy Daemon (SPD) is such a piece of software. It stores local security policies and provides an API through which other applications in the same workstation can use its services. We would prefer all workstations to have an SPD of some kind.

4.1 Functions

The SPD has several to some extent independent functions. It makes access control decisions and maintains secure network connections. It may help other protocols and applications in their security-related tasks. The SPD also stores local policy information in its database. Despite storing the information, the enforcing of a policy is not solely a task of the SPD. Applications have to co-operate and use the services SPD provides. A single policy can guide the behavior of many applications and it would be difficult for the SPD to explicitly control what all the other applications do.

The fundamental idea of access control is shown in Figure 6, which depicts the concept of a reference monitor. The reference monitor's input is a request from a subject and its output is an answer whether the service usage is granted or refused. If the service is refused, the monitor may also provide additional information about the reasons for this. The reference monitor concept shows the access controller as a block that contains all the information needed to make the decision, it does not need to consult the specific resource, any other software or a human administrator to produce the answer. If it does not have enough information, the answer is to forbid the service from the requestor.

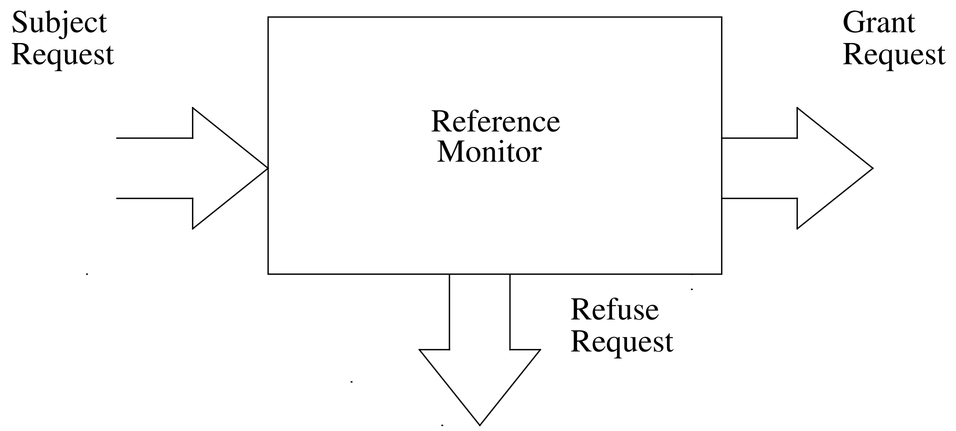

FIGURE 6. Reference monitor concept [AMO94]

The SPD is not quite this independent. Even if it has all the needed information concerning the requested resource, it may have to ask for help in understanding all components of the request. Besides answering access control requests, the SPD may need to create such requests or to parse and store other credential information. One of the resources to control is the policy database of the SPD itself: who can read or change what policies?

The function of maintaining network connections consists of establishing secure connections to other workstations, accepting connection requests and storing connection information. This may have to be done for the purposes of several different protocols. A secure connection between the parties is usually a prerequisite for any other communication, e.g. requesting a service. The protocols can also have other questions or mappings for the SPD to resolve.

Central to the network connections functionality is the concept of a Security Association (SA). It is a list of management parameters used by communicating entities to secure the communication. Security association data contains the results of a mutual agreement between hosts on keys, algorithms used and additional parameters concerning the use of the algorithms.

Generally, an SA can be bidirectional or unidirectional. We have adopted the one-way definition, meaning that two communicating peers have two SAs in use, one for each direction. The task of key management is to establish and update the variables in an SA, while the task of a security protocol is to use these variables for its purposes. [HOG97]

4.2 Environment

4.2.1 Software level

On the level of other software blocks, we can look at the surroundings of the SPD inside the server. There are several such blocks that may need to communicate with the SPD, for example IPSEC, ISAKMP and generic application programs.

Internet Protocol Security (IPSEC) [ATK95] is a part of the next generation network layer protocol IPv6. These are IETF proposals to overcome the problems in the current IPv4. IPSEC offers identification, integrity and confidentiality of IP datagrams. The protocol is independent of the used cryptographic algorithms. IPSEC uses ISAKMP for key management.

Internet Security Association and Key Management Protocol (ISAKMP) [ISAKMP97] is an application-layer protocol framework. It defines a set of procedures for identifying the communicating parties, creating and managing security associations, key generation techniques, and threat mitigation (e.g. denial of service and replay attacks) [ISAKMP97].

Oakley Key Determination Protocol is a protocol used by two communicating parties to agree on secure and secret keying material. The basic key exchange mechanism is based on the Diffie-Hellman key exchange algorithm, combined with RSA based identification. Oakley is designed to be a compatible component of ISAKMP. This means that Oakley data is carried in ISAKMP payloads. [ISAOAK97] defines more exactly how ISAKMP and Oakley are used together. [HOG97]

4.2.2 Network environment

Going outside of a single computer, we now focus on some of the different network entities. There are three possible entities assumed to actively participate in a normal access control situation. First, there is the client, who wants to use a certain service. Then there is the Server, which provides that service. Finally there is a policy server, which may assist the server in its access control decisions. There may, for example, be one such a trusted server in an organization. We will call the policy server as the Internet Policy Manager (IPM). Each of these entities can of course work in a different role in some other transaction.

Figure 7 shows these entities communicating over ISAKMP-based secure protocols. The picture shows how it is possible that the entities have different kind of versions of the SPD software, because of having somewhat different requirements for the SPD functionality. The SPD+ of IPM needs to be able to answer requests, which the server-SPD can not. The server-SPD may have chosen to handle only one certificate format, for example. The client can have either of these SPD versions. The client may be a normal workstation in an other organization, in which case it would probably have a normal SPD version, or it could be a stand-alone host, which uses no IPM-like services and has all the features of SPD+ installed.

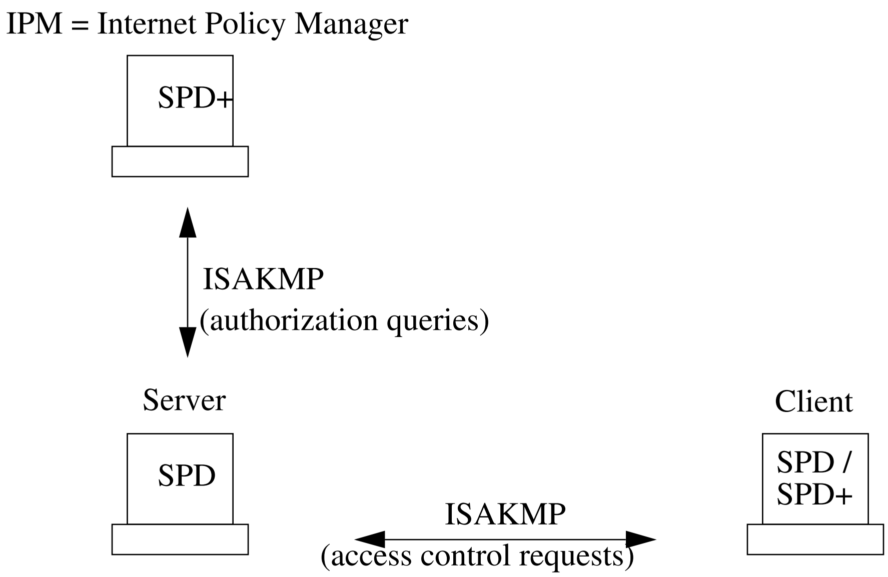

FIGURE 7. Different security infrastructure entities

4.3 Structure

Figure 8 shows a generic protocol layer security architecture. The structure of the SPD was not designed to look exactly like this, but was soon found to contain all the same components.

The control part is responsible for the operation of the whole SPD. It maintains the variables and uses all the mechanisms that the other parts provide for it. The decisions about whether a certain mechanism should be used are made by the control part based on the variables.

Variables are a place where the control part stores data. Security related information is stored in the security variables. Such information includes certificate cache, security association data and local policy information. The security association data determines the agreed ways of communication between hosts. The policy information is

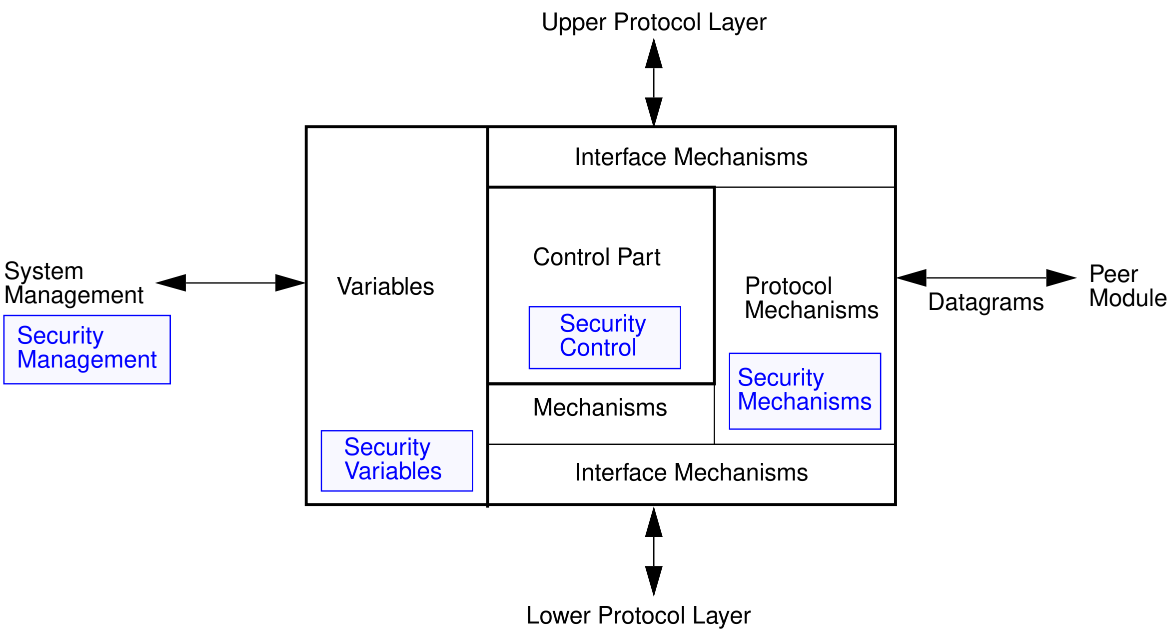

FIGURE 8. Generic protocol layer security architecture [KAR91].

restrictions on the use of the workstation and its allowed actions on the network. It consists of organization-wide security policies as well as user -level policies .

Protocol mec hanisms handle the communications tow ards other entities . The control part asks protocols to establish secure connections to other entities and protocol part returns the information regarding the newly established c hannel. Even if the communication to other entities is dra wn to the right in F igure 8, in practice it flows through the lower interface mec hanisms .

The interface mec hanisms provide the communication functions . The upper interface is tow ards applications that use the services of the SPD and the lower interface is to the TCP or UDP la yer of the protocol stac k.

Other mechanisms that are drawn below the control part, provide subsystems for the control part to use. The SPD needs at least certificate handling mechanisms, which verify or create certificates or certificate chains.

Systems management in the left of the picture corresponds to local configuration management of the SPD. For example, it is necessary to configure the trusted local key and possibly the key of the trusted IPM.

5 Implementation

The previous chapter presented a conceptual introduction to the structure of the SPD. In this chapter we take a look at a concrete implementation. Before going to the details like the object models, we introduce the programming languages, tools and other architectural elements involved.

5.1 Architectural Elements

5.1.1 General considerations

The designer of a communication protocol needs to consider all the things that can technically go wrong in a network. These are usually modeled as statistical bit-error rates, packet-loss rates and such. Some new protocols do not even try to react to all errors, because the network is seen to function reliably most of the time.

The designer of a secure communication protocol has to assume that the network is malicious and will make the worst possible errors or frauds at the worst possible moment. A cryptographic protocol must be able to securely recover from a conscious and well-organized attack.

The same kind of distinction applies to the implementor of 'ordinary' software and secure software. There are numerous examples to show us that an apparently innocent neglect, an unfortunate implementation choice, or a simple misunderstanding has undermined the security of an otherwise well-designed system.

5.1.2 Java

The Java programming language originated as part of a research project to develop advanced software for a wide variety of network devices and embedded systems at Sun Microsystems Inc. It is designed to be platform-independent, highly object-oriented and simple. It has been developed with C++ in mind; however, there are many differences such as automatic garbage collection, no header files, no multiple inheritance, and no pointer data types.

The platform independence is achieved by compiling the Java source code to bytecode, an architecture neutral intermediate format. Bytecode is interpreted at run-time by Java Virtual Machines in the different environments. After the source code is compiled to bytecode in some environment, this same bytecode should run in all processors and operating systems. There is also the possibility of including native, platform dependent code, but then the portability of the code is lost.

If the lower performance caused by interpretation cannot be tolerated, a JIT (Just In Time) environment can be used. A JIT compiler converts a piece of bytecode into native machine code before it is executed. Typically, all of the bytecode is not converted but each method is converted just before it is used for the first time. If some method is not used at all, it will not be converted. It is also assumed that there will soon exist hardware implementations of the Java Virtual Machine; that will also improve performance.

Java has built-in support for multithreading and synchronization. This enables the programmer to write programs in such a way that parallel processes do not conflict with each other when calling the same method. Multithread support is very useful in client-server type of network applications where several calls can arrive almost simultaneously.

Security is an integral part of Java in several ways. The Java Security Architecture makes it possible to control the resources a program is allowed to use. Different kind of restricted environments, sandboxes, can be made for different types of software. It is essential to be able to place the software blocks that are fetched over an untrusted network in a tight sandbox.

Another aspect of Java security is the Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE). JCA provides an implementation independent set of tools which help to work with cryptographic algorithms, keys and such. There are two different APIs in JCA; an API for the software developer who uses the cryptographic functions and another for the one who provides the cryptographic function implementations. JCE is an addition to JCA and contains the cryptography currently restricted by the export laws of the United States.

Java programs can include documentation in their source code, in special documentation comments. A program called Javadoc can then produce hypertext web pages automatically from the source code of the program and these documentation comments. This removes the need to separately write the same class and method information both in the code and in the documentation.

It is clear that Sun Microsystems Inc. wants to make profit with the Java concept in the long run. It is not clear whether that will conflict with the interests of the Java users. Java is not an open standard. Sun is openly publishing the Java specifications and also is in the process of standardizing them with ISO, but Sun is firmly going to keep the rights to the name Java. This will allow everyone to make perfectly compatible products, but not use the name Java without being a Sun licensee. Sun's own Java Development Kit (JDK) implementations are currently free of charge, but this may not be the case in the future.

5.1.3 Design patterns

Design patterns are simple and elegant solutions to specific problems in object-oriented software design. [GAMMA] describes a set of such patterns to start with. These pattern descriptions can widen our design vocabulary in an important way. Instead of describing the designs on the level of algorithms, data structures and classes, we can catch different aspects of the larger behavior with a single word. The benefits of using a pattern are greatly amplified at the stage of documentation or when discussing the design with another person who is familiar with patterns. [GAMMA] classifies the patterns it presents for three different purposes: creational, structural and behavioral patterns.

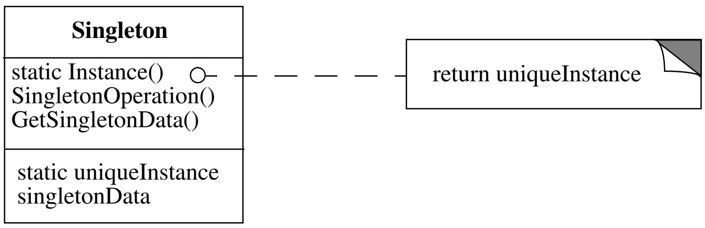

FIGURE 9. The Singleton Pattern Structure [GAMMA]

Creational patterns abstract the instantiation process. They help make the system independent of how its objects are created, composed and represented. Creational patterns give a lot of flexibility in what gets created, who creates it, how it gets created, and when. An example of a creational pattern is the Singleton pattern shown in Figure 9. This is one of the simplest patterns. It just ensures that a class only has one instance and provides a global point of access to it, a very useful and often needed feature.

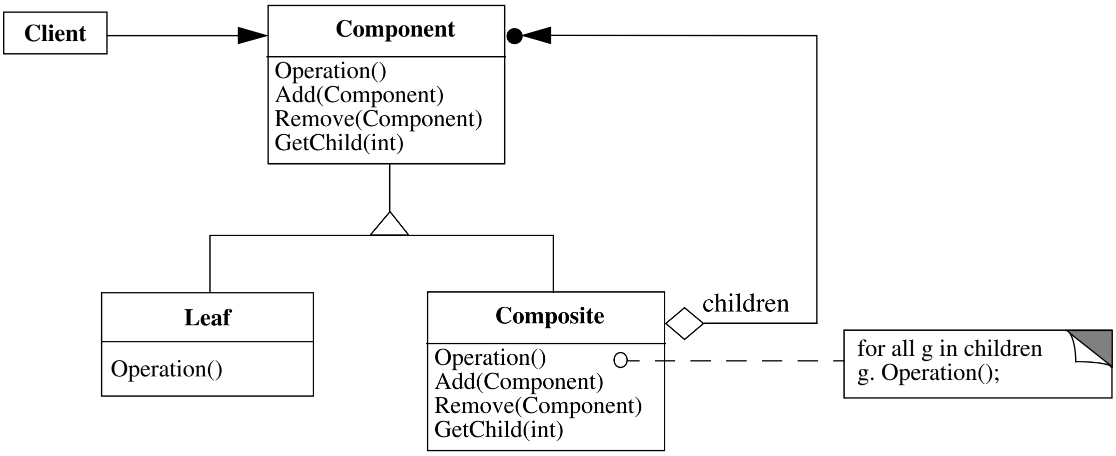

FIGURE 10. The Composite Pattern Structure [GAMMA]

Structural patterns are concerned with how classes and objects are composed to form larger structures. Class-based structural patterns rely on inheritance while object-based rely on object composition. The Composite pattern shown in Figure 10 is a structural pattern. It is a way to represent part-whole hierarchies. It enables clients to treat all objects in a composite tree in a uniform way.

Behavioral patterns are concerned with algorithms and the assignment of responsibilities between objects. They describe not just patterns of objects or classes but also the patterns of communication between them. An example of a behavioral pattern is the Visitor pattern in Figure 11. A visitor travels through an object structure performing some operation to the components. The operation is defined in the particular visitor and not in the component classes. This way we can define new operations by making a new visitor and without touching the complex object structure.

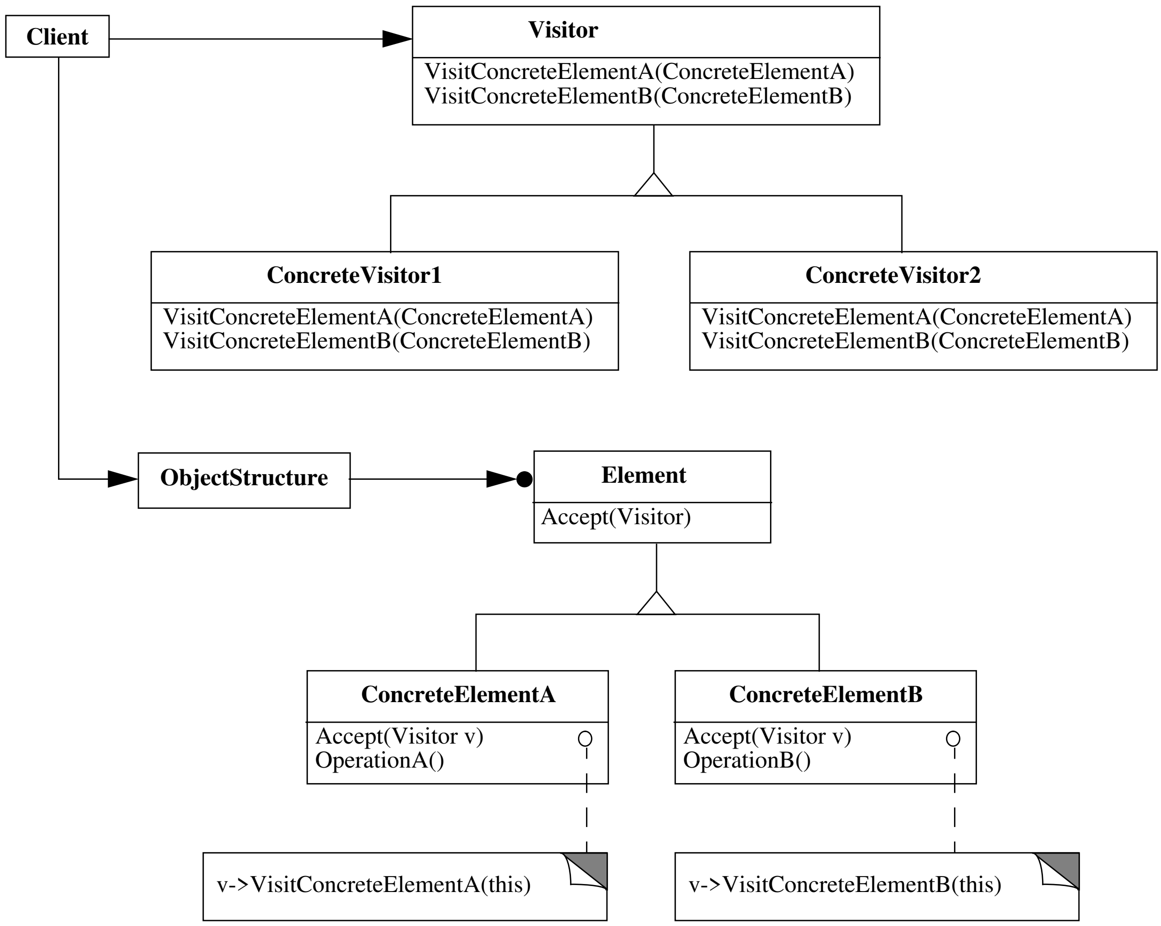

FIGURE 11. The Visitor Pattern Structure [GAMMA]

5.1.4 Previous work

This work is a follow-up to some previous implementations of security infrastructure components in the Telecommunication Software and Multimedia Laboratory of Helsinki University of Technology. M.Sc. Timo Aalto has written a STREAMS-based IPSEC prototype implementation for Solaris [AAL96]. M.Sc. Kaj Höglund has implemented an ISAKMP and Oakley key determination protocol prototype [HOG97]. Bengt Sahlin has made the IPSEC and IPv6 protocols using Conduits+ protocol framework [SAH97]. This framework was also used in the ISAKMP implementation. The SPD implementation described here will continue to build the system towards an authorization-based distributed security environment.

5.2 SPD Implementation

An object-oriented design tool called PepperSeed was taken into use early on in the project. It created the skeleton of the Java code from the system design and the rest of the code was filled in later. The comments written on the PepperSeed find their way to the code, and the class interface descriptions can be produced straight from the code with the Javadoc utility. PepperSeed is also able to hold several complementary views of the designs. The tool supports the Unified Modeling Language (UML). All the object models that follow are in the UML notation produced by PepperSeed.

Figure 12 shows the top-level class structure of the SPD. ControlPart is the main thread waiting for some tasks to appear to the TaskQueue. ControlPart and SecurityVariables have an obvious correspondence in Figure 8 on page 34. The Adapters are the interface mechanisms of the SPD and each of them is implemented as a separate thread. The certificate handling functions of CertData fall to the category of other mechanisms. The ISAKMP module, which is a protocol mechanism, is used through an adapter and is not shown in Figure 12. The properties of some classes are used to show that the Singleton design pattern is used.

There is one primary TaskQueue, which is filled by the three different adapters: the AppsAdapter, the IsakmpAdapter and the IpsecAdapter. The exact usage and output of the SPD depends on the interface created by the specific adapter. The queue is read by the ControlPart, which has two other queues for storing tasks not yet completed. Task has the following concrete subclasses: AnswerAccess-

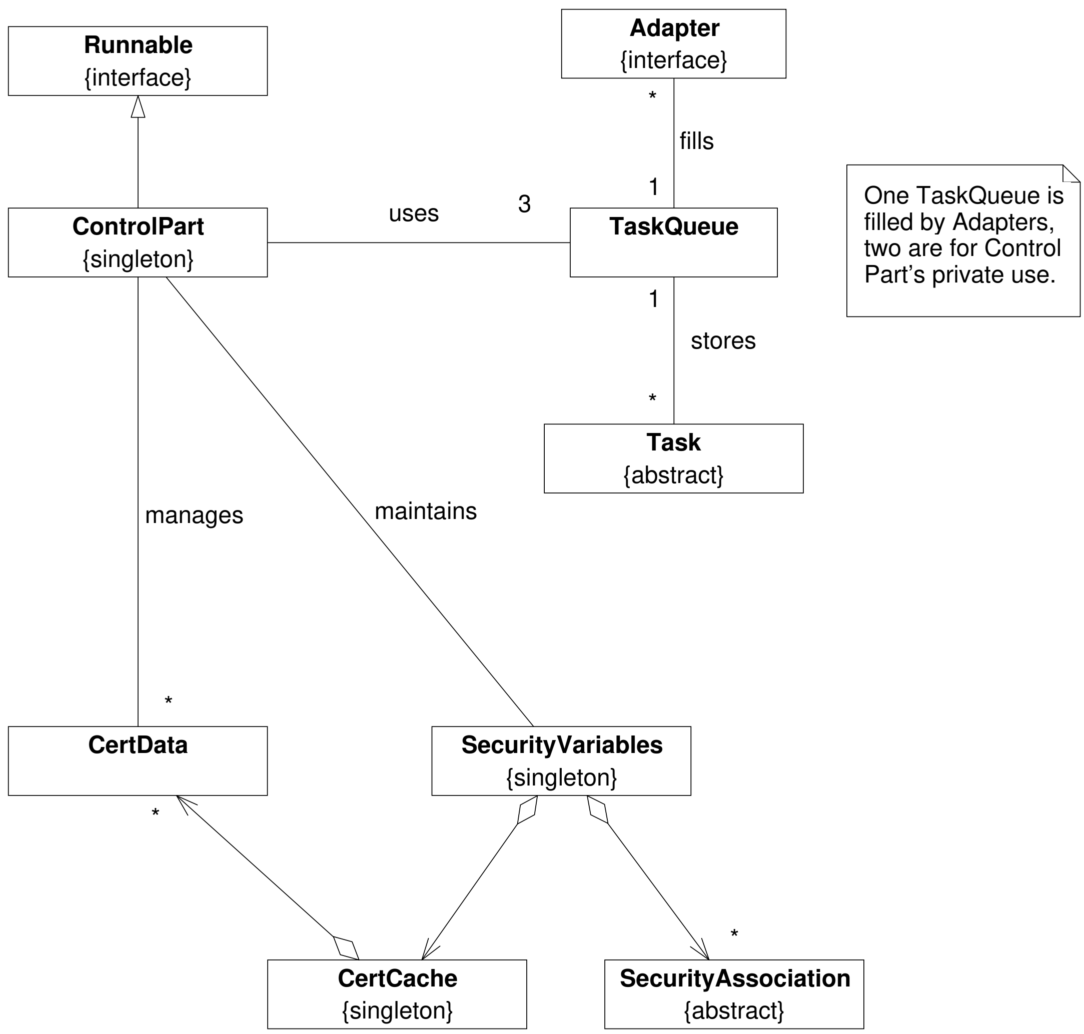

FIGURE 12. Top-level object model of the SPD

Request, ResolveSm, ResolveSpi, CreateNewSa, IpmAnswer and ChangeVariables.

CertData can contain a single certificate or a certificate chain. The class can do conversions between different certificate formats and reduce the chains. CertData may contain several formats at the same time. Certificates can be stored to the CertCache, which is a part of the SecurityVariables. Besides certificates and SecurityAssociations, it is possible to add any kind of data to the SecurityVariables.

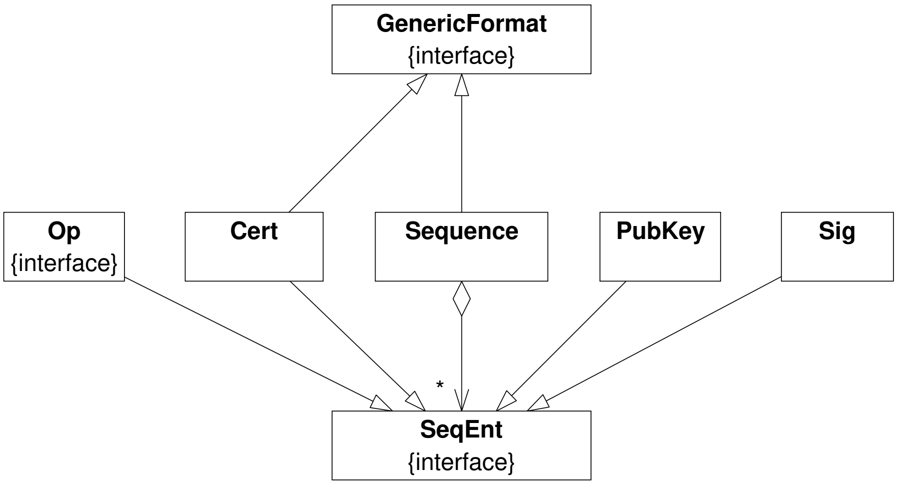

FIGURE 13. Sequence-structure of GenericFormat

5.3 SPKI Implementation

The most important of the certificate formats is the so-called Generic-Format. It is the internal format designed to contain several types of certificates. The structure of the GenericFormat is shown in Figure 13. A certificate sequence consists of certificates, public keys, signatures and operations.

The data of a single certificate is stored in a tree-like class structure, part of which is shown in Figure 14. The structure is implemented according to the Composite pattern. All of the classes in the structure are either Composites or ByteString-objects. Composites contain other Composites and ByteStrings.

The SPKI format is first stored in a class called CanonicalSpki. This is basically a plain byte array representation. The conversions between CanonicalSpki and GenericFormat are done using the Visitor pattern.

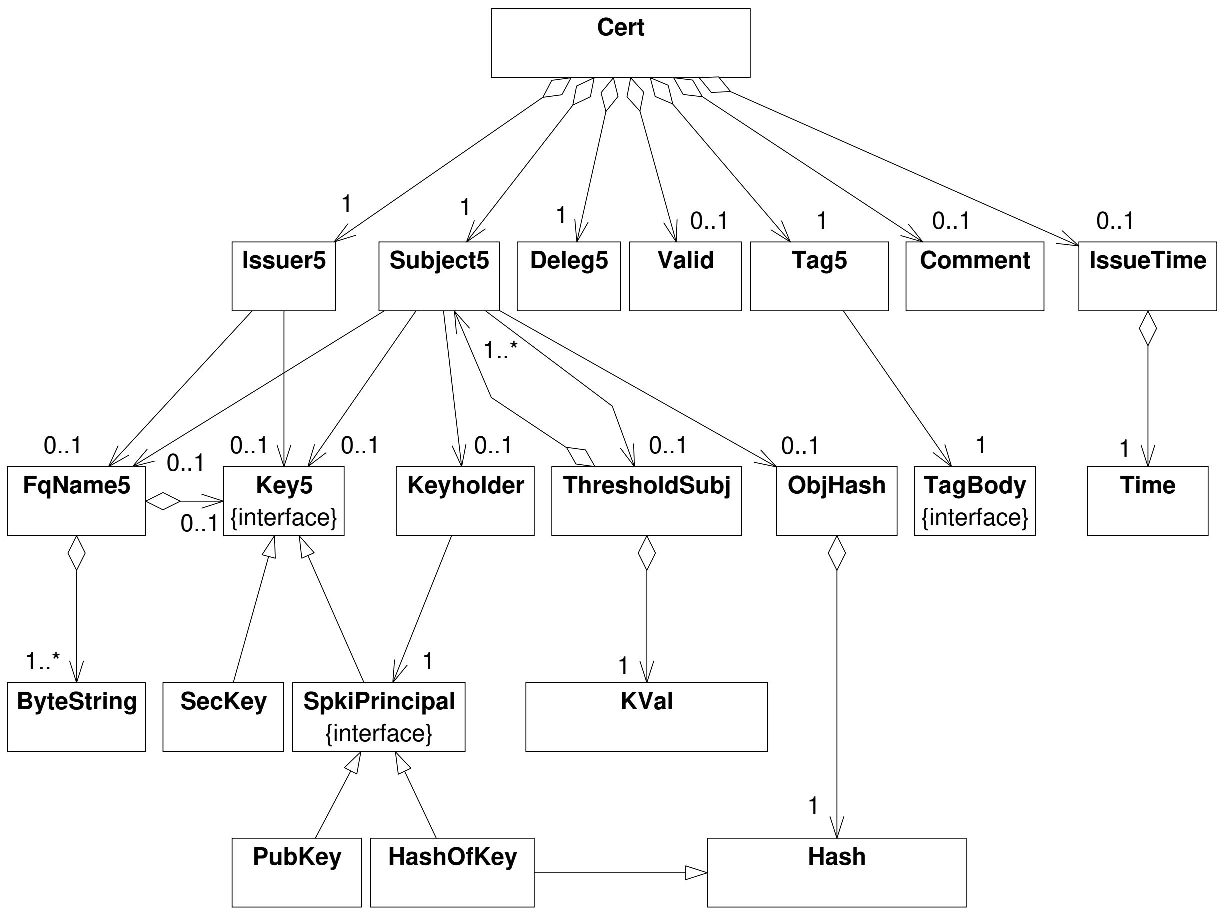

FIGURE 14. Fields of a certificate

From the certification definition subsets of SPKI, the certificate structure and the visitors support the full definition. SpkiMakerVisitor traverses GenericFormat and creates a new CanonicalSpki object on the way. SpkiParserVisitor interprets the CanonicalSpki byte array, creates new objects to the GenericFormat structure and travels through the structure it is creating.

6 Experiences and Conclusions

"Ah!" said Eeyore. "Lost your way?"
"We just came to see you," said Piglet. "And to see how your house was. Look, Pooh, it's still standing!"
"I know," said Eeyore. "Very odd. Somebody ought to have come down and pushed it over."
"We wondered whether the wind would blow it down," said Pooh.
"Ah, that's why nobody's bothered, I suppose. I thought perhaps they'd forgotten." [MIL28]

We do not want the Internet to look like Eeyore's house and rely to the hope that nobody bothers to push it down. We have to find out the possible security threats and use all the available means to strengthen the construct. IPSEC and IPv6 are going to incorporate security to the network layer of the protocol stack instead of leaving it purely as an application layer problem. This is not just a philosophical question about where the peer-to-peer security functions should be implemented. The routing and other functions of the whole stack need to be protected also. Now adays it is all too easy to spoof the routing or the name system and the consequences of this may be catastrophic.

We are going to have a public key infrastructure of some kind. In addition to this, we will need means by which entities are authorized to do something. Whether this functionality will be combined with the key infrastructure, like in the case of SPKI, or primarily be done with separate private Access Control List -like constructs, is still an open question. It is practically impossible to predict what the Internet Security Infrastructure will be like in ten years.

6.1 SPKI Suitability

The idea behind SPKI is still very immature. The possibilities and the restrictions of SPKI have not been explored in depth yet. The current drafts are very much in the state of development. In spite of these facts, the concept looks very promising. We hope that the results of the IETF working group will get enough publicity so that the critical mass of knowledge on these intricate subjects will be reached.

The name of the person is an essential fact in access control only if we happen to use a mechanism that binds the access rights to this name. This is almost never necessary. By binding the rights straight to the key, we get a simpler, more tailor-made system that has additional benefits, such as anonymity. Unless we want to hurry our journey towards the Orwellian society of no protection of intimacy, this is very important.

6.2 Java Experiences

No programming language can be the best choice to every application. The languages have different characteristics and are suited to different tasks. Among the languages, Java may be the first of a new breed. It has an interesting combination of properties not previously generally available. Platform independence at the level of compiled code has not been the favorite approach so far. The decreasing cost of computation has made this possible. We will see other languages using this feature in the future.

Java has obvious advantages for most programmers. No hassle with memory management and the debugging of it. An advanced library of classes for several areas of usage including networking, security, thread support and user interfaces. The possible weak points of Java are performance and standardization. However, the last word on both of these subjects is still not said.

6.3 Reaching the Goals of the Project

At the beginning of the project, some goals were stated. In the area of SPD implementation, the most important were: 1) to produce a functioning prototype based on the current SPKI drafts, 2) that the prototype has clear and modular architecture, and 3) to have a comprehensive documentation of the prototype.

What comes to the first goal, it is not obvious that is has been completely reached when this is written. The characteristics and features of the prototype have evolved as the work has progressed. The SPD has a lot of things to do and the implementation of all the different features is not yet completed, but the work continues in this area. Design patterns were used to help reaching the second goal. It is noteworthy to say that the implementor's understanding about what is meant by the modularity or the re-usability of the designs has increased during the project. The quality of the high-level architecture and the chosen designs should be on the required level. The chosen implementation language and the object-modeling tool have been of great help in reaching the third goal. The documentation has been completed hand in hand with the implementation.

Besides these three implementation goals, there was one more primary goal in the project: to produce the written part of a Master's Thesis. This objective was obviously reached.

7 References

[AAL96] Aalto, T. P., A Unix Streams Implementation of the Internet Protocol Security, Helsinki University of Technology, 1996.

[AMO94] Amoroso, E., Fundamentals of Computer Security Technology, McGraw-Hill, 1994.

[AN95] Anderson, R., Needham, R., Robustness principles for public key protocols, Crypto'95, 1995.

[ATK95] Atkinson, R., Security Architecture for Internet Protocol, RFC 1825, Naval Research Laboratory, 1995.

[BBK94] Beth, T., Borcherding, M., Klein, B., Valuation of Trust in Open Networks, University of Karlsruhe, 1994.

[BLA96] Blaze, M., Feigenbaum, J., Lacy, J., Decentralized Trust Management, Proceedings IEEE Conference on Security and Privacy, 1996.

[CHA97] Chadwick, D., Young, A., Merging and Extending the PGP and PEM Trust Models - The ICE-TEL Trust Model, IEEE Network Magazine, May/June 1997.

[ELL96a] Ellison, C., Establishing Identity Without Certification Authorities, Proceedings USENIX Security Symposium, 1996.

[ELL96b] Ellison, C., Generalized Certificates, <URL: http://www.clark.net/pub/cme/html/cert.html>, 1996.

[GAMMA] Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[HOG97] Höglund, K., A Conduits+ and Java Implementation of the Internet Key Management Protocol, Helsinki University of Technology, 1997.

[ISAKMP97] Maughan, D., Schertler, M., Schneider, M., Turner, J., Internet Security Association and Key Management Protocol (ISAKMP), version 8, draft-ietf-ipsec-isakmp-08.{ps, txt}, 1997.

[ISAOAK97] Harkins, D., Carrel, D., The resolution of ISAKMP with Oakley, version 3, draft-ietf-ipsec-isakmp-oakley-03.txt, 1997.

[KAR91] Karila, A., Open Systems Security - an Architectural Framework, Helsinki University of Technology, 1991.

[LANDAU] Landau, C., "Security in a Secure Capability-Based System", Operating Systems Review, pp. 2-4, October 1989.

[MIL28] Milne, A., A., The House at Pooh Corner, Methuen Children's Books, 1928.

[NIK] Nikander, P., Modelling of Cryptographic Protocols - A Concurrency Perspective, Helsinki University of Technology, 1997.

[PGP] Zimmermann, P., The Official PGP Users Guide, MIT Press, 1995.

[IPKI97] Housley, R., Ford, W., Polk, W., Solo, D., Internet Public Key Infrastructure, Part I: X.509 Certificate and CRL Profile, Internet Draft, draft-ietf-pkix-ipki-part1-05.txt, 1997.

[SAH97] Sahlin, B., A Conduits+ and Java Implementation of Internet Protocol Security and Internet Protocol, version 6, Helsinki University of Technology, 1997.

[SDSI96] Rivest, R., Lampson, B., SDSI - A Simple Distributed Security Infrastructure, 1996.

[SPKI97] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T., Simple Public Key Certificate, Internet Draft, draft-ietf-spki-cert-structure-02.txt, 1997.

[STA95] Stallings, W., Network and Internetwork Security Principles and Practice, Prentice Hall, New Jersey, 1995.

[X509] International Telegraph and Telephone Consultative Committee (CCITT), Recommendation X.509, The Directory - Authentication Framework, CCITT Blue Book, Vol VIII.8, pp. 48-81, 1988.

[YKB93] Yahalom, R., Klein, B., Beth, T., Trust Relationships in Secure Systems - A Distributed Authentication Perspective, Proceedings of the IEEE Conference on Research in Security and Privacy, 1993.