Certifying Trust

Ilari Lehti[1], Pekka Nikander[1]

Helsinki University of Technology, Department of Computer Science,
FI-02015 TKK, Espoo, Finland
{ilari.lehti, pekka.nikander}@hut.fi

Abstract. A basic function of all signatures, digital or not, is to express trust and authority, explicit or implied. This is especially the case with digital signatures used in certificates. In this paper, we study the trust relationships expressed by the certificates used in X.509, PGP and SPKI. Especially, we present and revise the idea of a certificate loop, or a loop of certificates from the verifying party to the communicating peer, requesting access or acceptance. We also show how that kind of certificate loops can be used to explicitly express security policy decisions. In the end of the paper, we briefly describe our own SPKI implementation that is specially tailored towards policy management. The implementation is based on Java and build using Design Patterns. It functions as a separate process, providing security services to the local kernel and applications.

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 [10], 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) [14] and on our implementation of a Policy Manager based on that proposal. Our system allows trust and authority to be explicitly represented in the form of certificates.

Suppose Pooh would use such a system. After deciding of 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 push 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 Trust Models

Trust is a belief that an entity behaves in a certain way. Trust to a 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 [16]. Yet another refinement might be to define a degree of trust needed towards the other entities [3]. 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.

1.2 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 im-

portant information when some outsider is trying to decide whether an organization can be trusted in some respect. This is one situation where it is of use to define the policy in a systematic manner, 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.

1.3 Digital Certificates

A certificate is a signed statement about the properties of some entity. In a digital certificate the signature is a number 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 rest 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. In the case of an identity certificate, this is 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.

1.4 Certificate Loops

The idea of certificate loops is a central one in analyzing trust. The source of trust is almost always the checking party itself. For example, if a user wants to authenticate that a networked server is, actually, providing the service the user wants to use, the cer-

tificates used for this check must be trusted by the user. Specifically, the first certificate in a certificate chain must be trusted, implicitly or explicitly. Similarly, when the server wants to check the user's access rights, the certificates used to authenticate the user must be trusted by the server. Again, the server must be configured to trust the certificates in the chain.

Thus, a chain of certificates, typically implicitly starting at the verifying party and ending at the party claiming authority, forms an open arc. This arc is closed into loop by the online authentication protocol where the claimant proves possession of its private key to the verifying party. Such a loop is called a certificate loop. In Section 4.1, we return to this issue in more detail.

1.5 Outline of This Paper

In the next section we will discuss the currently most popular certification systems and compare the different certification approaches. In section 3 we are going to take a closer look to the SPKI and some of the new ideas behind it. Section 4 shows our view of certificate loops and presents parts of our implementation. In section 5, we will discuss the possible future directions of Internet security. The last section sums up our key ideas.

2 Expressing Trust With Certificates

In this section we describe some concrete certificate infrastructure proposals and compare them with respect to trust. These systems divide 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 hierarchy.

2.1 Certifying Identity

We did briefly 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.

Fig. 1 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 mathematics of cryptography. In reality the strength depends on the trust relationships involved and is always subjective. We could draw the private key and the public key separately. This would bring one more step to the 'loop'. The binding between these two keys is the strongest binding of all.

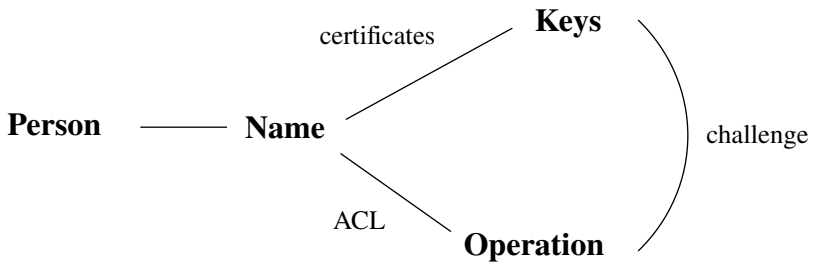

Fig. 1. Identification certificate bindings

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 [14]. 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?") [4]. 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 [11] has been a success since its introduction. X.509 [15], while not taking an actual flying start, has lately gained some popularity despite the lack of any working global scale X.500 directory service.

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. [4]

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. [4]

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. [4]

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 [12] nor the current usage of the system gives any support for such a practice, though.

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 [7]. 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 [8]. A controlled and secure directory service would be possible to create, but apparently there is currently no large market demand for it.

Trust models. The PGP users can trust anyone they want; all users are equal. This kind of freedom will cause a "web of trust" to be created between the users. In addition to trusting a certain key as valid, it is possible to define the degree of trust to a person as an introducer. Each individual creates, certifies and distributes their own keys. PGP rejects the concept of official certifying authorities being more trustworthy than the guy/girl next door.

One of the problems in the X.509 trust hierarchy is that it has a centralized trusted entity as a root that everyone with their differing needs should be able 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.

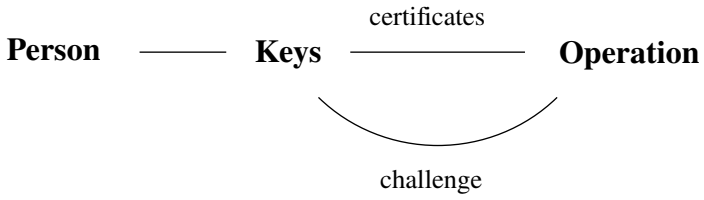

Fig. 2. Authorization certificate bindings

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. 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 [5]. The ICE-TEL calls its trust model as 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.

2.2 Certifying Authorization

The bindings of an authorization certificate system are shown in Fig. 2. 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. 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.

So, 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.

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 applica-

tions, 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, [6] discusses different possibilities how to bind persons to keys without certification authorities. [14] 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 [4], [7]. 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 [13], SPKI and PolicyMaker [4]. In section 3, we will have a more detailed discussion about SPKI.

Trust models. 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 [1], 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.

3 Simple Public Key Certificate

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. [14]

3.1 Principals and Naming

The SPKI principals are keys. Delegations are made to a key, not to the keyholder. 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 refer to a set 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 one, if the intention is to change the binding to a new. 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 [9] 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. [14]

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.

3.2 Certificate Format

The current SPKI proposal uses 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 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. 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.

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 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.

3.3 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 [14]:

(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. 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 [14].

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.

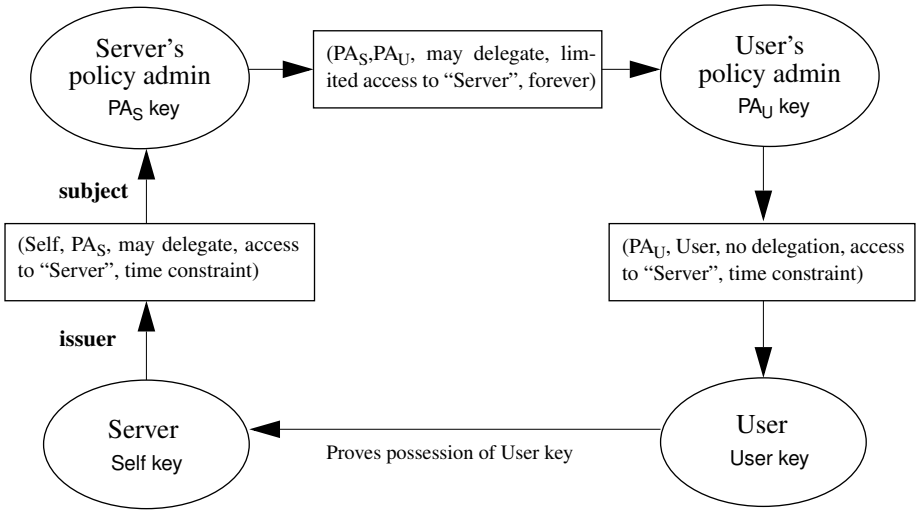

Fig. 3. Basic authorization certificate loop

4 Implementation

We have created a prototype of an SPKI based policy manager. The prototype is written in Java, using the Design Pattern structures [8] in order to promote software reuse and build on best known practices. The purpose of the implementation is to facilitate real life tests with policy based certificates and management.

Before going to the details, we take a look at a typical certificate usage and then introduce some of the architectural elements involved.

4.1 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.

Fig. 3 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 delegations. The permission propagates through the policy administrators and their certificates to the service user.

To be complete, this example needs also another loop. Fig. 4 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.

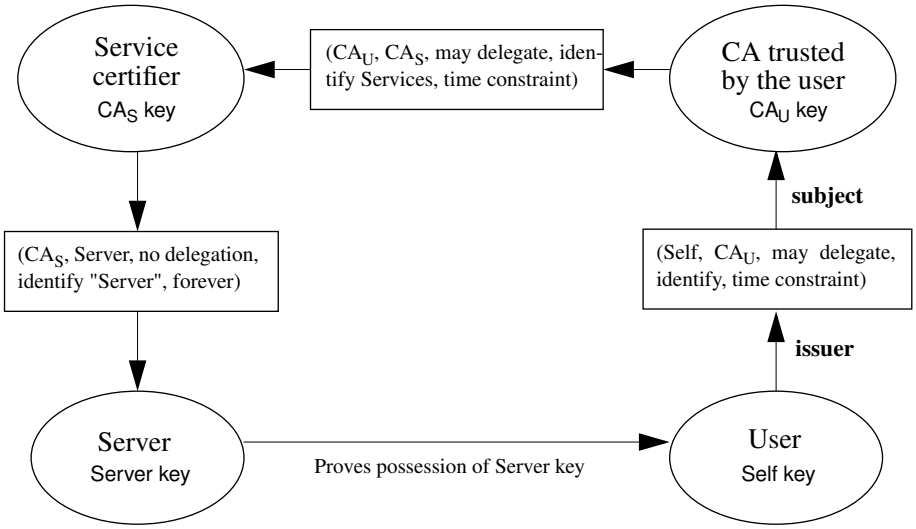

Fig. 4. Basic service identification loop

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 certificates, 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 Reduction Certificate (CRC), which the server may use as part of the final reduction. One reason for using CRCs may be that the full chain contains certificate formats which the server does not understand. The server may also make its own CRCs for performance reasons, so that it need not make the same reduction several times.

4.2 Design Patterns

Design patterns are simple and elegant solutions to specific problems in object-oriented software design. [8] 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. [8] classifies patterns for three different purposes: creational, structural and behavioral patterns.

4.3 Policy Manager Implementation

Our prototype consists of a main module and a number of protocol adapters. The latter ones interface the policy manager to various security protocols such as IPSEC and ISAKMP. We plan to add support for additional security protocols later on.

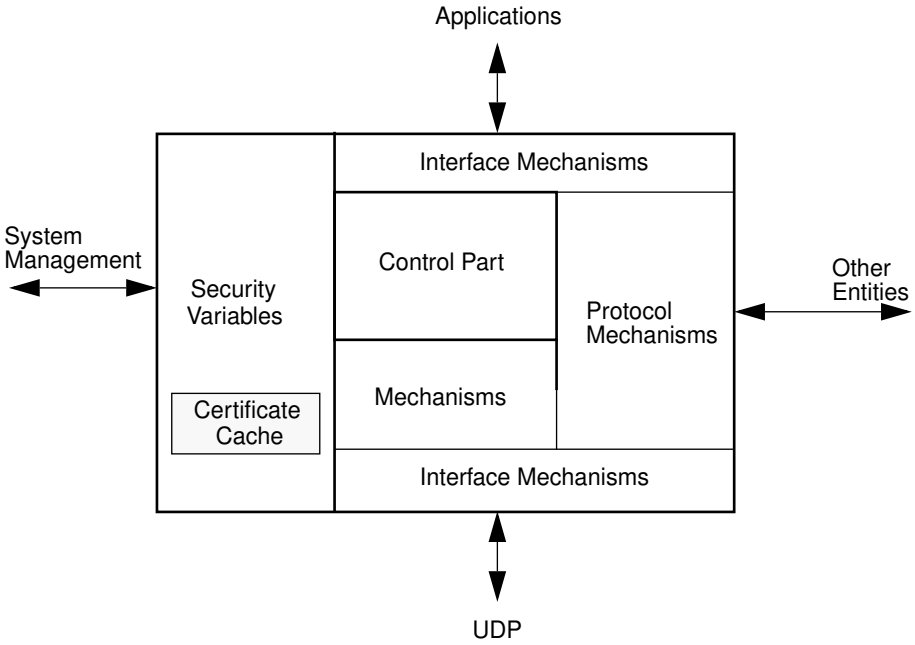

Fig. 5. The structure and connections of Policy Manager

The basic structure of the Policy Manager is shown in Fig. 5. It has a control part, which is the main thread waiting for some tasks to appear to the task queue. The interface mechanisms are called adapters and each of them is implemented as a separate thread. The primary task queue is filled by the three different adapters: the applications adapter, the ISAKMP-adapter and the IPSEC-adapter. The exact usage and output of the Policy Manager depends on the interface created by the specific adapter.

The certificate handling subsection of the main module reads in chains of SPKI certificates, reduces them on demand, and participates on ISAKMP based authentication protocols. The result is a highly configurable ISAKMP security association policy that allows the properties of the created associations to be set up according to the restrictions and limitations expressed with the certificates.

In addition to handling certificate data, the system makes access control decisions, maintains secure network connections and stores policy information. It may also help other protocols and applications in their security-related tasks.

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 Policy Manager to resolve.

The service provider may use another trusted policy server, which may assist it in access control decisions. There may, for example, be one such a trusted server in an organization. Even if the service provider has all the needed information concerning the

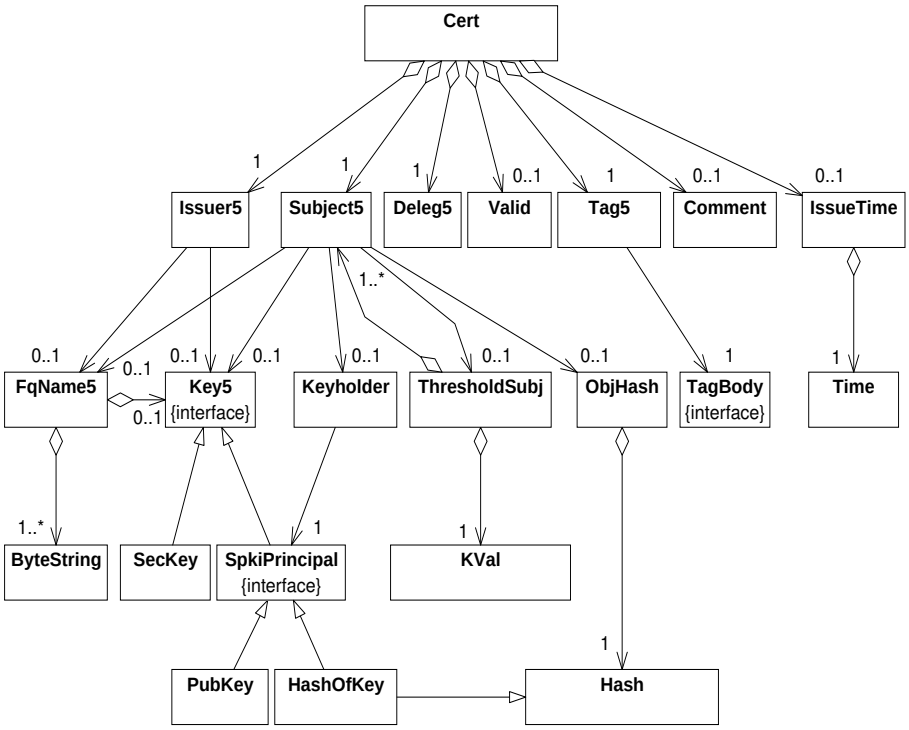

Fig. 6. Fields of a certificate

requested resource, it may have to ask for help in understanding all components of the request. Besides answering access control requests, the Policy Manager may need to create such requests or to parse and store other credential information. One of the resources to control is the policy database that it maintains: who can read or change what policies?

4.4 SPKI Implementation

We have designed an internal format for certificates. This format can hold the certificate information from several different transfer formats, for example PGP, X.509 and SPKI. This generic format is largely based on the SPKI structure and can therefore also contain and reduce certificate sequences. Certificate data can be stored to a certificate cache, which is a part of the trusted security variables of the system.

The data of a single certificate is stored in a tree-like class structure, part of which is shown in Fig. 6. The structure is implemented according to the Composite pattern, which is a way to represent part-whole hierarchies in a tree structure. It enables clients to treat all objects in the composite tree uniformly. All of the classes in the structure are either composites or byte strings. Composites contain other composites and byte strings.

Arriving certificate data is first stored in an instance of a class specific to that format. The conversions between specific formats and generic format are done using the

Visitor pattern. 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.

5 Future Directions

"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."

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 [2] 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. Nowadays 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 cryptographic 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 Conclusions

Digital certificates can be interpreted as expressions of trust. From this viewpoint, certifying user identity is pretty meaningless. Winnie the Pooh doesn't benefit much from the information that his new friend's name is Tiger according to Piglet, however true and trusted this piece of information was. In addition to that, Piglet must tell him how trustworthy Tiger is (or, alternatively, how trustworthy tigers are in general).

Thus, in order to successfully express trust — and thereby security policy constraints — we have to add semantic meaning to the certificates. SPKI, and its cousins SDSI and PolicyMaker are initial steps on this path. The idea behind SPKI is still very immature. Its possibilities and restrictions have not been explored in depth. 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.

References

1. Anderson, R., Needham, R.: Robustness principles for public key protocols, In Proceeings of Crypto'95, 1995.
2. Atkinson, R.: Security Architecture for Internet Protocol, RFC 1825, Naval Research Laboratory, 1995.
3. Beth, T., Borcherding, M., Klein, B.: Valuation of Trust in Open Networks, University of Karlsruhe, 1994.
4. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized Trust Management, In Proceedings of the IEEE Conference on Security and Privacy, 1996.
5. Chadwick, D., Young, A.: Merging and Extending the PGP and PEM Trust Models - The ICE-TEL Trust Model, IEEE Network Magazine, May/June, 1997.
6. Ellison, C.: Establishing Identity Without Certification Authorities, In Proceedings of the USENIX Security Symposium, 1996.
7. Ellison, C.: Generalized Certificates, http://www.clark.net/pub/cme/html/cert.html.
8. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
9. Karila, A.: Open Systems Security - an Architectural Framework, dissertation, Helsinki University of Technology, 1991.
10. Landau, C.: Security in a Secure Capability-Based System, Operating Systems Review, pp. 2-4, October 1989.
11. Milne, A. A.: Winnie-the-Pooh, The House at Pooh Corner, Methuen Children's Books, 1928.
12. Zimmermann, P.: The Official PGP Users Guide, MIT Press, 1995.
13. Housley, R., Ford, W., Polk, W., Solo, D.: Internet Public Key Infrastructure, Part I: X.509 Certificate and CRL Profile, draft-ietf-pkix-ipki-part1-05.txt, 1997.
14. Rivest, R., Lampson, B.: SDSI - A Simple Distributed Security Infrastructure, 1996.
15. 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.
16. International Telegraph and Telephone Consultative Committee (CCITT): Recommendation X.509, The Directory - Authentication Framework, CCITT Blue Book, Vol VIII.8, pp. 48-81, 1988.
17. Yahalom, R., Klein, B., Beth, T.: Trust Relationships in Secure Systems - A Distributed Authentication Perspective, In Proceedings of the IEEE Conference on Research in Security and Privacy, 1993.