Distributed Access-Rights Management with Delegation Certificates

Tuomas Aura

Abstract. New key-oriented discretionary access control systems are based on delegation of access rights with public-key certificates. This paper explains the basic idea of delegation certificates in abstract terms and discusses their advantages and limitations. We emphasize decentralization of authority and operations. The discussion is based mostly on the SPKI certificates but we avoid touching implementation details. We also describe how threshold and conditional certificates can add flexibility to the system. Examples are given of access control between intelligent networks services.

1 Introduction

New distributed discretionary access control mechanisms such as SPKI [14] and PolicyMaker [8,9] aim for decentralization of authority and management operations. They do not rely on a trusted computing base (TCB) like traditional distributed access control [22]. Instead, the participants are assumed to be untrusted the way computers on open networks (e.g. Internet) are in reality.

The decentralization, however, does not mean sliding back to anarchy such as the PGP web of trust [26]. The new systems offer ways of building local relations and setting up local authorities that arise from the personal and business connections of the participants. The access control mechanisms do not mandate any hierarchical or fixed domain structure like, for example, Kerberos [18] and DSSA [16]. All entities are equally entitled to distribute rights to the services in their control and to act as an authority for those who depend on them for the services.

The main mechanism used in the new access control systems is delegation of access rights with signed certificates. The signing is done with public-key cryptography. With a certificate, one cryptographic key delegates some of its authority to another key. The certificates can form a complicated network that reflects the underlying relations between the owners of the private signature keys.

By taking the cryptographic keys as their principal entities, the systems avoid dependence on trusted name and key services such as the X.500 directory [12]. If any names are used, they are not global distinguished names but relative to the users [1,24].

This paper explains the principles behind the delegation certificates in an abstract setting without exposing the reader to implementation details. The discussion is based primarily on the SPKI draft standard although we will not touch

Appeared in Secure Internet Programming (LNCS 1603).

certificate formats. We stress distribution, scalability and locality of policy decisions. Application examples are given from the Calypso service architecture for intelligent networks [19]. The emphasis is this paper is on practical issues. More theoretical treatments of distributed access control can be found in [2,4]. Some support from implementations is in [3,20]. Code libraries and products using SPKI are expected to appear when the standardization work nears completion.

Sec. 2 introduces delegation certificates an explains how they aid decentralization. Sec. 3 describes two enhancements, threshold schemes and validity conditions, that add flexibility to the basic certificates. The limitations of the certificate-based approach and ways to overcome them are the topic of Sec. 4. Sec. 5 concludes the paper.

2 Access Control with Delegation Certificates

Sec. 2.1 introduces delegation certificates and some basic concepts. In Sec. 2.2, we describe first how access rights are distributed and verified with certificates. Then we consider redelegation and extend the verification procedure to cover chains of delegation certificates. Sec. 2.3 explains why certificates are an appealing alternative for other methods of distributed access control.

2.1 Basic Delegation Certificates

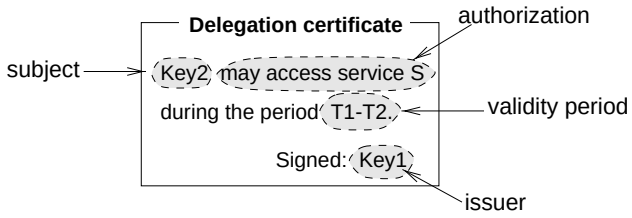

Fig. 1. With a delegation certificate, the issuer shares authority with the subject.

A delegation certificate (Fig. 1) is a signed message with which an entity grants access rights that it has to another entity. We are interested in systems where the certificates are signed with public-key cryptography and the entities granting and receiving access rights are, with some exceptions, cryptographic keys. A certificate has the meaning:

$$S_K(\text{During the validity period } T_1 - T_2\text{, if I have any of the rights } R,\\ \text{I give them also to } K'.)$$

($S_k(\ldots)$ denotes a signed message that includes both the signature and the original message.) The key that signed the certificate (K) is the *issuer* and

the key to whom the rights are given (K') is the *subject* of the certificate, and the rights R given by the certificate are the *authorization* (following the SPKI terminology). With the certificate, the issuer delegates the rights to the subject.

All delegation certificates have a *validity period* ($T_1 - T_2$) specified on them. When the certificate expires, the subject loses the rights that the certificate may have given to it. Together with the authorization field, this parameter is used for regulating the amount of trust the issuer places on the subject. Extremely short validity periods are used to force on-line connections to the issuer. (For simplicity, we often omit the validity period in the text below.)

The authorization is usually the right to use certain services. Sometimes, it can be an attribute that the subject uses as a credential to acquire access rights. Such attributes can be interpreted as abstract rights that may not directly entitle the subject to any services but help in acquiring such rights. The syntax of the authorization is application dependent and each application must provide its own rules for comparing and combining authorizations. (See [14] Sec. Examples for some typical authorizations.)

Some characteristics of delegation certificates are that any key can issue certificates, a key may delegate rights that it does not yet have but hopes to obtain, and the issuer itself does not lose the rights it gives to the subject. In the following, we will discuss these and other properties of delegation in detail.

Our view of the world is *key-oriented*. The entities possessing, delegating and receiving access rights are cryptographic key pairs. The public key is used to identify the key pair and to verify signatures. The private key can sign messages. It is held secret by some physical entity that uses the key and the rights attributed to the key at its will. All keys are generated locally by their owners. There is no limit on the number of keys one physical entity may own. On the other hand, if a physical entity is to receive any rights from others, it must be represented by at least one key. Most public-key infrastructures (e.g. X.509) are *identity-oriented*. In them, access rights are given to names of entities and the names are separately bound to keys.

Delegation certificates also differ from traditional access control schemes in that any key may issue certificates. There is no central or trusted authority that could control the flow of access rights. All keys are free to delegate access to services in their control.

The delegation takes effect only to the extent that the issuer of a certificate itself has the authority it is trying to delegate. Nevertheless, it is perfectly legal to issue delegation certificates for rights that one does not yet have or for broader rights than one has in the hope that the issuer may later obtain these rights. Sometimes a key may delegate *all* of its rights to another key. The policy for calculating the access rights received by the the subject when the issuer itself does not possess all of the rights listed in the certificate depends on the type of authorizations in question. In this paper, we consider only *set-type* authorizations as most literature [4,14]. The subject gets the intersection of the rights held by the issuer and the rights mentioned in the certificate. Since this limitation is

true for all delegation certificates, it is usually not explicitly mentioned. If we also ignore the validity period, the certificate above can be written, in short,

$$S_K(K' \text{ has the rights } R.)$$

Since the signing of a certificate happens locally at the physical entity possessing the private issuer key, the act of signing does not invalidate any existing certificates or affect existing rights. The subject of the certificate gets new rights but the issuer does not lose any.

In this way, delegation is less powerful than transfer of rights where the originating entity loses what it gives. On the other hand, delegation is far easier to implement. The simplicity of the implementation (signing a certificate) in a distributed environment is the reason why delegation is preferable to transfer as an atomic access control primitive. (See Sec. 4.1 about implementing transfer.)

2.2 Certificate Chains

This section shows how the certificates are used as a proof of authority and how the rights can be passed forward through several keys and certificates.

We begin by considering the simple case of access right verification where the owner of a service has issued a certificate directly to the user of the service (like K delegates to the public key K' above). When the user wants to use its rights, it signs a request with its private key (K') or in some other way authenticates (e.g. by establishing an authenticated session) the access request with the private key. This can be construed as redelegating the rights to the request. The user attaches the delegation certificate to the access request and sends both to the server.

Every service platform has either a single master public key that controls all access to it or access control lists (ACL) that determine the privileged public keys for each service. When the server receives an access request with an attached delegation certificate, it first verifies that the certificate is signed by a key controlling the requested service (K). It then checks that the authorization in the certificate is for the service and that the key requesting the service is the same as the subject of the certificate (K'). In this scenario, the certificate behaves like a capability. The signature protects the capability from falsification and binds it to the subject key.

Just as a key may delegate rights to services it directly controls, it may also redelegate rights it received by delegation from other keys. In this paper we assume that redelegation is always allowed unless a certificate explicitly forbids it.

When a key delegates to another key and this key in turn redelegates to a third one, and so on, the delegation certificates form a chain. In the chain, the access rights flow from issuers through certificates to subjects. The original issuer is usually the service producer and the final subject is a client of the service.

If all the certificates in the chain delegate the same access rights and specify the same validity period, these rights are passed all the way from the first issuer

to the subject of the last certificate. But it is not necessary for the certificates to have the same authorization field and validity period. We remember that the rights obtained by the subject key are the intersection of the rights possessed by the issuer and the authorization field of the certificate. Consequently, the rights passed through the chain of certificates can be computed by taking the intersection of the rights possessed by the first issuer with the authorizations on all the certificates in the chain. (Sometimes the intersection can be empty meaning that no rights are passed all the way.) Likewise, the validity period of the of the chain is the intersection of the periods specified on the individual certificates. For example, in the chain of Fig. 2, *Key4* receives the right to the web pages "http://S/file" for today only from key *Key1*.

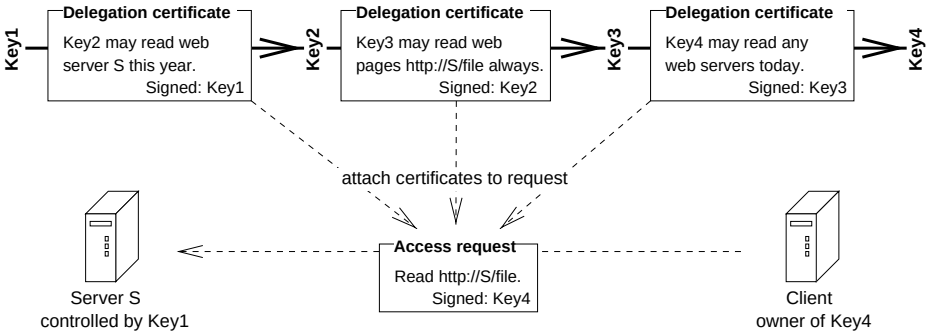

Fig. 2. A chain of delegation certificates

Since the certificates can be issued by anyone to anyone, they do not necessarily form simple chains. Instead, the certificates form a graph structure called *delegation network*. In the delegation network, there may be many chains of certificates between the same pair of keys. Naturally, the rights passed between two keys are the union of the rights passed by all individual chains between them. (Set-type authorizations are combined with the union operator. Other policies are possible for other types of authorizations.)

When a key requests a service and it has obtained the access rights through a chain of certificates, it attaches the entire chain to its request (Fig. 2). The server will verify that the chain originates from a key controlling access to the requested service, that it ends to the key making the request, that each certificate in the chain is signed by the subject of the previous certificate, and that each certificate in the chain authorizes the request. Several chains of certificates can be attached if the combined rights delegated by them are needed for the access.

The signing of an access request can also be thought of as redelegation to the request. Furthermore, the request may be program code. In that case, it is natural to think that the last key in a chain redelegates to the code and the code makes the actual requests. Consequently, delegation certificates should be used

to express this delegation. This is done so that the last key signs a delegation certificate where the subject is not a key but a hash value of the program code. We will see an example of this in Sec. 2.4.

In a way, the delegation certificates behave like signed requests for capability propagation (see e.g [17]). The requests are honored only if the signer itself has the capability. However, it is not necessary for the server or for a trusted party to process each propagation before the next one is made, and no new capabilities need to be produced before the rights are used. Instead, the information is stored in the form of the delegation certificates.

Although the most obvious way of managing certificates is to accumulate them along the chain of delegation and to attach them to the service requests, there is no compelling reason to do this. The certificates can be stored and managed anywhere as long as the verifier gets them in the end. The accumulation is not always even possible if the certificates are not issued and renewed in the order of their positions in a particular chain.

Managing long chains of certificates can become a burden. A technique called *certificate reduction* saves work in verifying the certificate chains. We observe that two certificates forming a chain

$$S_{K_1}(K_2 \text{ has the rights } R_1.) \text{ and } S_{K_2}(K_3 \text{ has the rights } R_2.)$$

imply a direct delegation from K_1 to K_3:

$$S_{K_1}(K_3 \text{ has the rights } R_1 \cap R_2.)$$

The third certificate is redundant and signing it will have no effect of the access rights of any entity. Hence, K_1 can sign it after verifying the chain of signatures. Continuing the process inductively, a chain of any length can be reduced into a single certificate. If the reduced certificate is used repeatedly as a proof of access rights, savings in the verification time can be substantial. In order to compute the intersection of the authorizations, the issuer of the reduced certificate must have some understanding of the contents of the authorization fields. Other than that, the reduction is a mechanical procedure that does not involve any decision making. Certificate reduction is the main technique of certificate management in SPKI.

2.3 Distribution of Authority and Operations

This section explains the rationale behind delegation with certificates. We emphasize suitability for open distributed systems with no globally accepted authority. The advantages center around the high level of distribution achieved both in authority and management work load.

The first key to the distribution is that the entities are represented by their cryptographic keys instead of names. Names are a natural way of specifying entities for humans but they are less suitable for cryptographically secure authentication. Ellison [13] discusses the complicated connection between keys and identities. In a key-oriented world, we don't need trusted third parties to certify

the binding between a name and a public key. Instead, the public key is used directly to specify an entity. Thus, the centralized or hierarchical certification authorities (CA) that are the heart of traditional identity-based access control lose their role in key-oriented systems. This is a major security advantage. For example in X.509, the name certificates come from a global hierarchy of officials (CAs) who must all be trusted with respect to any access control decision whose security depends on the correct mapping between a key and a name. Key-oriented access control avoids such obvious single points of failure. Fig. 3 illustrates the difference. It becomes even more obvious when we remember that, in a truly name-oriented system, the mapping from name to key must be done also for the service owner and for every key in a chain of delegation.

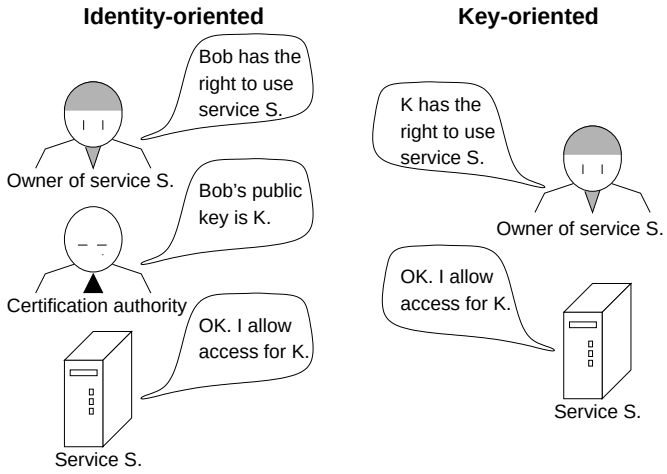

Fig. 3. In an identity-oriented system, security of all access control depends on trusted authorities. Key-oriented systems avoid this.

The omission of names also results in savings in communication and certificate processing. There is no need to contact a CA. The verification of the certificates in a chain is a straight-forward, mechanical procedure that involves no communication or policy decisions.

Sometimes we, nevertheless, need bindings between names and keys. This is usually because of the need to refer to legal persons or because names are a natural form of human input. In such cases, we must either trust name certificates from a some CA or, preferably, resort to SDSI-type linked name spaces [1,24] whose relativity is explicit. But when feasible, the most robust practice is to pass the correct public key from the subject to the issuer at the time when a delegation certificate is created and to issue the certificate directly to the key.

The effect of delegation on the distribution of authority, however, is greater than only eliminating the CA. Delegation-based access control does not con-

ceptually differentiate between keys that are allowed to grant access and ones that only use services. From technical point of view, all keys are equal regardless of the importance of rights they handle. Any key may issue certificates to others and distribute access rights to the services in its control without asking permission from any other entity.

The bottom-up formation of policy is the most distinguishing property of certificate-based access control. The certificates are issued locally by the entities that produce the services and, therefore, should be responsible for granting access to them. The certificates are formal documents of local trust relationships that arise from voluntary personal, technical and business relations between the entities possessing private keys. Nothing is mandated by global authorities. In a chain of delegation certificates, every link is a result of a local policy decision. Because of redelegation, the local decisions by individual key owners have global consequences.

The lack of enforced hierarchical structure makes the system open and scalable. Setting up new a new entity is as simple as generating a signature key. New keys may be created locally as new physical entities or services are introduced. In comparison, most traditional access control systems achieve scalability with a hierarchy of trusted entities or domains and require meta-level maintenance operations for changes in the hierarchy [12,16,18].

An important observation in the certificate management is that the storage and distribution of certificates is a separate concern from their meaning [8,9]. The integrity of the certificate data is protected by the signatures. Thus, untrusted entities can be allowed to handle them. Often, organizations will want to set up certificate databases for access-right acquisition and management. Similarly, most application software packages are unlikely to include their own certificate management. Most of the tasks can be done for them by untrusted servers or helper software. Nikander and Viljanen [23] describe a way of storing certificates in the Internet domain name service and a discovery algorithm based on [3].

The certificates should be considered as sets or graphs rather than as chains. This is because the order of issuance of the certificates does not necessarily have any correlation to their order in a particular chain. The time of issuance and the validity periods depend on the local trust relations behind the certificates. These are independent of whatever chains may be formed globally. If one certificate in a chain expires, only that one needs to be refreshed immediately. The other certificates in the chain remain valid. In this way, the system is distributed not only in space but also in time. It is even possible that the private keys or the owners of the keys who issued some of the certificates have ceased to exist by the time the certificates are used as a part of a chain. This is commonly so when temporary key pairs are used for anonymity or when a old system delegates its tasks and rights to a replacing one.

Certificate reduction allows a trade-off between communication and certificate processing cost. Reduction requires one to contact the issuer of the reduced certificate. This means that the entity must be on-line at the time of the re-

duction and that this on-line system must be secure enough to hold the private signature key. Luckily, the reduction engine can be fairly simple to build. If all the certificates in the chain delegate the same rights or if the application has straightforward rules for computing intersections of the authorizations, the reduction is a purely syntactical transformation and it can be fully automated. The issuer of the reduced certificate does not need to fully understand the meaning of the certificates. The reduction may save significantly in the costs of certificate transfer and verification.

Unlike mandatory access control mechanisms, delegation does not have a central reference monitor to supervise access and distribution of the access rights. Neither is there a trusted computing base (TCB) to monitor the actions of the distributed entities. This is because there is no global policy to enforce on the parts of the system. In open environments like the Internet, it would not be possible to implement any global controls. Instead, the policy is determined locally by the parts.

Delegation leaves more to the discretion of the participants than many other DAC mechanisms. The system does not enforce any policies to protect users from bad decisions. All the verifier of a certificate sees is public keys and signatures. It has no way of telling if the private keys belong to legitimate entities in the system. The certificate issuer at the time of signing should, of course, have a solid reason for trusting the subject with the delegated rights. But the systems leaves it to each key owner to judge by itself who can be trusted. The issuer may or may not know the name or identity of the subject key owner. Moreover, redelegation creates a new degree of freedom. When redelegation is allowed, anyone can share his rights with others. This is equal to universal grant rights from anyone to anyone.

It may seem that redelegation should be controlled. It is, indeed, possible to add conditions on the certificates limiting redelegation. In SPKI, the choices are to allow or forbid redelegation completely. A certificate that forbids redelegation can only be the last certificate in a certificate chain.

There are, however, appealing arguments for allowing free redelegation. It may be convenient for the client to authorize someone else to use the rights on its behalf, or the client may want to redelegate the rights to one of its subsidiaries. The internal organization of the clients of a service should not be a concern to the service providers. Free redelegation makes the internals of the system parts more independent thus furthering distribution.

The key-oriented nature of the system also obscures the semantical meaning of the restrictions on redelegation. Only in special circumstances does the issuer of a certificate know that the private part of the subject key is held permanently secret by a certain physical entity. In many cases, the issuer accepts any key given by the entity that is to receive the rights. In general, there is no guarantee that the corresponding private key will not be revealed to others. Giving out the key would spread the rights as effectively as redelegation but, unlike in redelegation, the rights and their validity period could not be limited. Usually, there are also

other ways to redistribute the services without the agreement of the originator, e.g. establishing proxy servers and outright duplication of the server data.

Instead of forbidding redelegation, the issuer of a certificate should consider changing the authorization and validity period. Minimizing the scope of delegated rights is the most natural way to express limited trust in the subject. The certificates make it easy to reconsider the rights and validity in each step of a delegation chain.

All in all, delegation does well the part of access control that is easy to implement: maximally discretionary distribution of access rights. Mechanisms for identity certification, limits on redistribution, rights transfer and revocation can be added where they are required. However, such features in principle require a more complex infrastructure with a TCB, tamper-resistant modules, trusted third parties or on-line communication. Therefore, the basic access control system should not require their use. We believe that there are many instances where pure delegation is a sufficient mechanism and corresponds well to the real-life access-control needs.

2.4 Access Control for IN Services

Delegation certificates are most suitable for use in distributed systems with no globally accepted security policy or authority and in open systems with no central registration of the servers and clients. They are unlikely to find applications in high-security environments where mandatory access control and trusted systems are a rule. Although most applications will be on the Internet, we will look at an example from the telecommunications world.

Calypso [19] is a distributed service architecture for intelligent networks (IN). It is designed for ATM access networks where a workstation running the Calypso service platform controls an ATM switch. Calypso provides flexible distribution of service and network control functions among service clients, servers and network nodes. The same architecture could also reside on top of other types of network equipment such as IP switches and firewalls.

Calypso is based on a business model where the network operator who owns the infrastructure offers network resources to service providers (SP). These resources are the lowest level of Calypso services. Service providers can either market the right to use their services to end users or they can sell them to other SPs for reselling or for use as building blocks of more sophisticated services. Complex services and their components form tree-like structures (Fig. 4).

All Calypso services are implemented as Java packages. A service may use other services by calling methods of the classes that belong to them. Hence, the network nodes must have an access control system that facilitates execution of code from mutually distrusting SPs and contracting of services between them. The Calypso security requirements and a tentative architecture for satisfying them were outlined in [6].

The IN access control mechanisms should encourage free formation of business relations between service producers and merchants distributing access rights

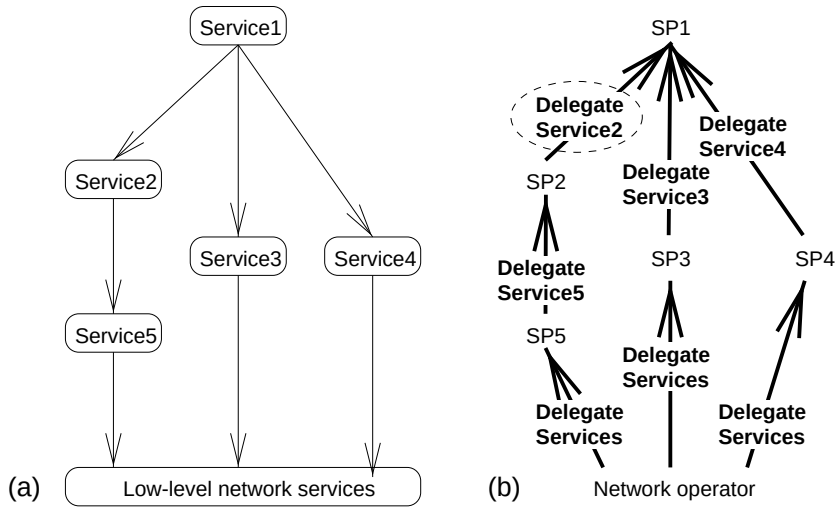

Fig. 4. Calypso service composition and delegation between SPs

to the services. Delegation, in a natural way, allows complex relations between the entities without forcing too much predefined structure on the service market.

Calypso uses SPKI certificates to delegate access rights between SP keys. The Calypso system differs from the standard scenario in that the last certificate in the chain always delegates the rights to (a hash value of) a Java package implementing the service that will use the access rights.

Fig. 5 shows how a service provider delegates access rights for its service to another SP that in turn delegates them to its service code. (This is the circled delegation step in Fig. 4(b).) When the code package is installed into the network nodes, it is accompanied by the certificates. In Fig. 6, the access rights are delegated through a service broker adding another link to the chain of certificates.

The delegated authorizations are always rights to access a Calypso service. Since services in the Calypso architecture are encapsulated into Java classes and packages, this gives a natural level of granularity for the access control.

Each service has exactly one key associated with it: the key of the service author or owner. This key distributes the access rights by issuing delegation certificates. The decision to delegate depends on the relations between the entities possessing the private keys. However, the service producers do not actually need to think in terms of access control policies. Instead, they sell and buy the certificates with business relations in mind.

The network operator or a service provider may want to review the client service implementation before granting access. This is because the access control mechanisms cannot effectively prevent denial of service, bad publicity and many other potential problems created by malicious or low quality services. The review

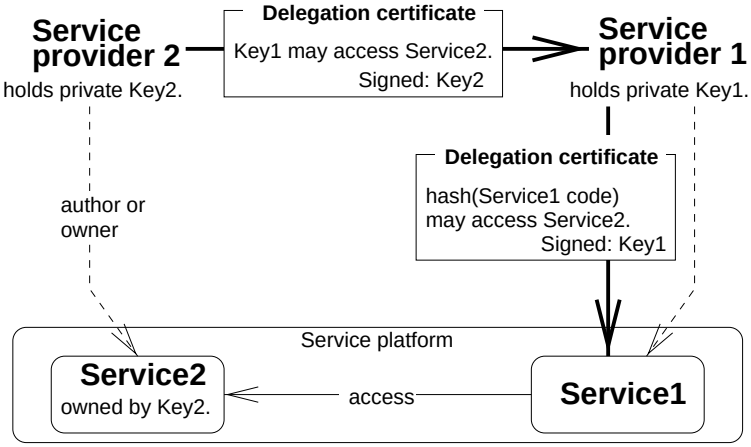

Fig. 5. Delegation to another service provider (SP) in Calypso

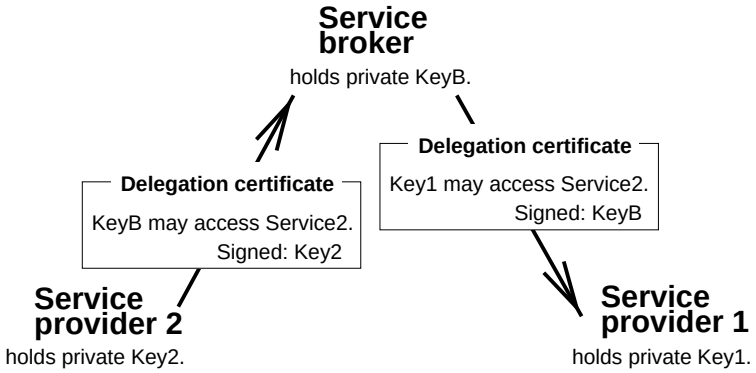

Fig. 6. Delegation through a service broker in Calypso

process is not easy to organize because the aim is to allow fast development of services by a large number of independent SPs. A possible solution is to use independent quality-control (QC) units that certify services if they meet some minimum quality criteria. In Fig. 7, the network operator grants access rights to SP1's code only after receiving the review results. If the QC writes a certificate to *Key1* instead, that means it has reviewed the production process of SP1 and the network operator can trust SP1 to do its own quality control.

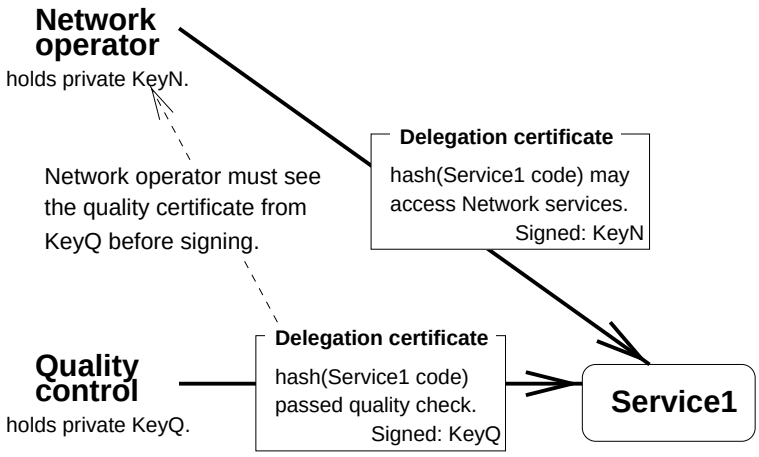

Fig. 7. Code quality control for IN services with basic delegation certificates

The example reveals a major limitation of the basic delegation certificates. The quality check must be passed before the network operator makes its decision. If changes are made to the code, the network operator must renew its certificate. But the network operator clearly does not make any new policy decision at that time. It only follows a fixed rule: if the quality check is ok, it will sign. In the following sections we will see how the requirement for quality check and other rules for deriving new access rights from old ones can be encoded into certificates.

3 Threshold and Conditional Delegation

The following sections introduce certificates with more complex structure than the ones we have seen so far. The enhancements increase significantly the flexibility of certificates as an access control tool. Threshold certificates are a means of dividing authority. Instead of giving the rights to a single subject, they are given to a group of subjects who must co-operate to use the rights. Sec. 3.1 describes the certificates and Sec. 3.2 explains how they are used. Conditional delegation is a way of expressing simple access control policy rules in certificates.

We will introduce the new type of certificates in Sec. 3.3 and look at applications in Sec. 3.4.

3.1 Threshold Certificates

Threshold certificates are an extension of the basic certificate structure. Instead of having only a single subject, they divide the authorization between several subjects. In order to use the rights given by a certificate, the subjects must co-operate. The certificate has a threshold number that determines how many of the subjects must agree.

A (k, n)-threshold certificate is a signed message like the following:

$$S_K(\text{During the validity period } T_1 - T_2, \text{ if I have any of the rights } R,\\ \text{I give them also to } \{K_1, K_2, ..., K_n\},\\ k \text{ of whom must co-operate to use or redelegate the rights.})$$

The usual way for the subjects to co-operate is to redelegate the rights to one of them or to some other single entity. In Fig. 8, $KeyC$ receives the right R from $KeyS$ because two subjects of the $(2, 3)$-threshold certificate co-operate to pass it to $KeyC$. When $KeyC$ wants to use the right R, it must attach all three certificates to its access request. It should be noted that the subjects of the threshold certificate do not need to delegate directly to the same key as in the figure. The delegation could go through independent or partially dependent chains of certificates and even through other threshold certificates before the shares are accumulated to a single key. The general structure of such networks was studied in [4].

The threshold certificates can also be used in some situations where there is no real threshold trust scheme. An example below will show how they may improve distribution and flexibility of the system.

(k, k)-threshold certificates where all subjects are required to co-operate are sometimes called *joint-delegation* certificates. *Open threshold certificates* [4] are a variation of the threshold certificates where each subject is given a separate certificate and new subjects can be added later.

3.2 Threshold Certificates and IN Access Control

There may be intelligent network services controlled by a group of SPs so that two or more are needed to grant access to the services. Such conditions can be directly expressed with threshold certificates from the service master key to the service providers' keys. A single SP may also want to ease the burden of storing its private key securely by distributing the authority between several keys. This is exactly what is pictured in Fig. 8. The SP generates three new keys K_1, K_2 and K_3. It then signs a the following kind of certificate with its master key K.

$$S_K(\text{Any 2 of } \{K_1, K_2, K_3\} \text{ have all my rights.})$$

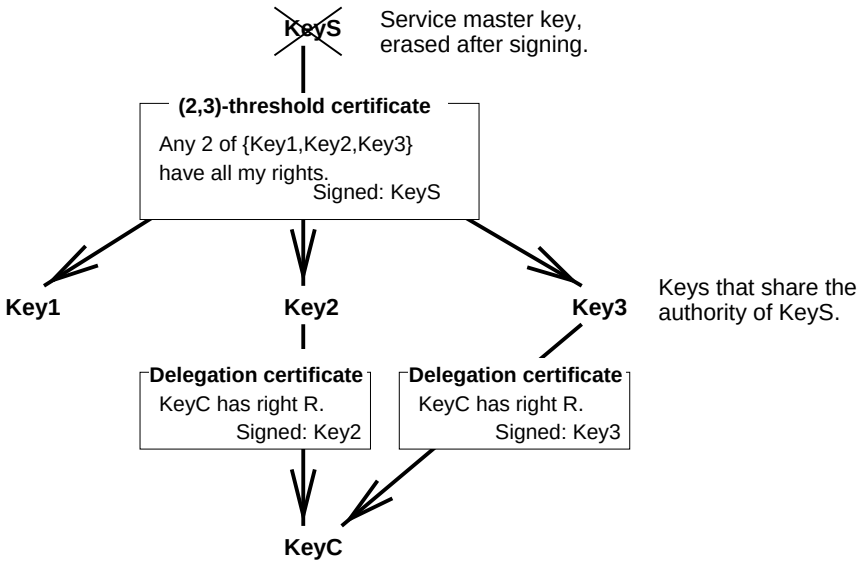

Fig. 8. A threshold certificate ($KeyC$ gets right R from $KeyS$)

This certificate allows any two of the new keys to operate on behalf of the original master key. The new private keys are stored in separate places while the old private master key is destroyed or stored in a safe place and never accessed. This protects both against theft and accidental loss of the private master key. If one share is compromised, it alone is not enough to misuse the service, and if one share of the three is lost, the other two can still grant access to the service. The SP can still advertise its old public key (K or $KeyS$) and receive new rights delegated to that key. The threshold certificate passes all these rights to the share keys who can authorize others. This kind of protection of private keys may prove to be a much more common reason to use threshold certificates than actual threshold trust schemes between business associates.

There is another, rather unexpected, application for the threshold certificates. We will see that flexibility can be added to systems like that of Fig. 7 by encoding an implication rule in a certificate. If Fig. 9, the quality-control key ($KeyQ$) certifies the code by granting the code *all* its rights. The network operator issues a (2,2)-threshold certificate for SP1 and QC. Thus, SP1 needs the agreement of the QC before it can use the network services. The three certificates together convey the right to access Network services from $KeyN$ to Service1. With these certificates, Service1 can prove its access rights when installed into a network node.

The QC key is never used for any other purpose than for certifying entities that have passed the quality check. A quality certificate can be issued to code or to service providers whose own quality control has passed an audit. In the

quality certificates, the QC delegates all its rights so that it does not need to know what kind access the quality certificates are used for. This means that the QC key should never be given any other rights than one share of a threshold certificate for the purposes of a quality check on whoever gets the other share.

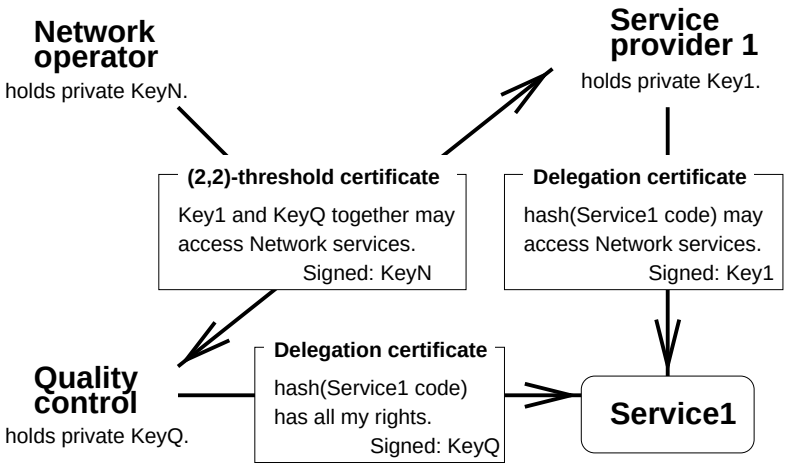

Fig. 9. Code quality control for IN services with threshold certificates

Compared to the use of basic delegation certificates (Fig. 7), the threshold certificate has the advantage that dependences between the quality control and the granting of access to the services have been reduced to minimum. It is not necessary to involve the network operator every time code is changed and the certificates can be signed in any order. The three certificates can be issued and renewed independently of each other.

Admittedly, this is a somewhat inelegant way to use threshold certificates. The rights delegated by the certificate are not encoded in the the authorization part ("all my rights") but in the signing key ("*KeyQ* is the quality-control key"). If there are several different authorizations (e.g. several types of quality check), the QC unit must have an equal number of signature keys. Moreover, comparing different levels of authorization becomes difficult when the authorizations are encoded in the keys. We observe that the threshold certificate in Fig. 9, in effect, carries the meaning

> "*Key1* may access the network services if it also has a quality certificate from *KeyQ*."

In section 3.4, we will introduce a new type of certificate that explicitly includes such conditions. The reason why we have described in length how to encode the same meaning into a threshold certificate is that only threshold delegation is currently supported by SPKI.

3.3 Conditional Certificates

Conditional certificates are like the basic delegation certificates except that they state additional conditions that have to be satisfied before the certificate is considered valid. A conditional certificate is a signed message with the following contents:

$$S_K(\text{During the validity period } P, \text{ if I have any of the rights } R,$$
$$\text{I give them also to } K' \text{ if it also has}$$
$$\text{the right } R_1 \text{ from } K_1, \text{ and}$$
$$\text{the right } R_2 \text{ from } K_2, \text{ and}$$
$$\text{the right } R_3 \text{ from } K_3, \ldots)$$

The certificate gives the rights to the subject only if all the conditions in the list are satisfied. The conditions always take the same form: they require the subject key of the certificate to have a certain authorization from a certain key. This is natural because any attribute that the subject may have can be verified only if it is expressed as an attribute certificate from a proper authority.

In order to use the certificate, the subject must provide a proof that the conditions are fulfilled. It does this by attaching appropriate certificate chains, one for each condition. The certificates in the proof of access rights form a tree (or a directed acyclic graph) rather than a chain.

When certain attributes are required before granting access to a service, a conditional certificate offers two advantages compared to the basic delegation certificates. First, the certificate is an unambiguous, standard-form statement of what kind of attribute certificates are still needed for the access. Secondly, the conditional certificate can be signed before obtaining the attribute certificates. Without conditional certificates, the client in need of access rights would first contact the issuer to find out what are the prerequisites, then try to acquire them and, in the end, return to the issuer with the collection of credentials in order to get the new certificate. When the decision rule is encoded in a certificate, the issuer needs to be contacted only once. The client may obtain the attributes before or after this contact. Thus, communication and synchronization between the entities is greatly reduced.

The conditional certificates express simple policy rules but they are, by no means, a general language for defining policies. For example, the certificates presented here cannot express symbolic rules. A more general language for expressing conditions, policies and limits on redelegation is a topic of active research.

3.4 Conditional Certificates and IN Access Control

Conditional certificates are just the right tool for the kind of situations where we slightly abused threshold certificates in Sec. 3.2. The code quality check for an IN service can be expressed as a condition. Fig. 10 reformulates the certificates of Fig. 9 with conditions. The result is functionally the same but the system is much more intuitive for a designer or an observer.

With the threshold certificates, the quality certificate told only indirectly what kind of authority it gives to the subject. The authorization was encoded in the signature key. In addition to being cumbersome to understand, this encoding has the disadvantage that the authorizations are not comparable. The conditional certificates, on the other hand, explicitly state the rights or attributes in the authorization field of the certificate and the authorizations can be compared. For example, we can let the IN Network operator write the required *level* of quality in the conditional certificate. If the QC issues a quality certificate with the same or higher level to Service1, the code will get the access rights.

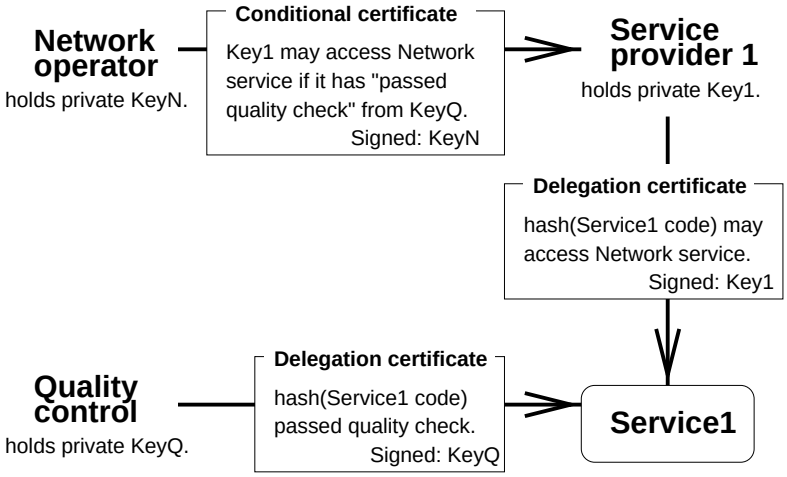

Fig. 10. Code quality control made simple with conditional certificates

4 Limitations

The flexibility of the certificates does not come completely without a price. There are many security goals that require centralized control and cannot be realized only with signed messages. We will consider such goals and see what kind of central or trusted services they imply. Sec. 4.1 discusses policies that require additional infrastructure. Sec. 4.2 brings up the issue of quantitative rights. In Sec. 4.3 we consider revocation and in Sec. 4.4 anonymity and auditing.

4.1 Expressive Power

The certificates are a form of discretionary access control. They cannot express mandatory policies like the Bell-LaPadula model [7] because we do not assume

any mechanism for enforcing a policy globally. A mandatory policy would require all equipment on the system to be under the control of some authority so that they can be trusted to follow the policy.

Another limitation is that the certificates can only convey policies where the rights of the entities grow monotonically as they acquire new certificates. It is impossible to verify that someone does *not* have a certificate. Consequently, separation-of-duty policies like the Chinese Wall policy [11] cannot be expressed with only certificates. They need some mechanism for keeping track of the previously granted rights. Moreover, if several distributed issuers give out certificates for different conflicting rights, these issuers must share a single view of the subjects' histories. The histories must be updated in real time when new certificates are issued. An equally difficult problem is that in a key-oriented system, one physical entity might use several keys to gain conflicting rights. We must first identify the entities whose duties we want to separate and then find a way of mapping keys to unique identities. For example, a trusted official could certify the keys to be unique personal keys of the participating persons. Altogether, separation of duty appears to be one of the greatest challenges for certificate-based access control.

A related problem is the separation of access rights and grant rights. In a key-oriented architecture, someone with only grant rights could easily subvert the protection mechanism by issuing the rights to a key held by himself [15]. Therefore, every key in practice has the rights that it is allowed to delegate to others. Like the separation of duty, separate policies for granting and using rights cannot be securely implemented unless each entity has a unique identity and keys owned by the entity are bound to the identity. Consequently, key-oriented systems usually do not even try to implement pure grant rights.

An occasionally needed access control feature is a proxy that can issue certificates with longer life-times than its own existence. For example, a manager on a vacation should be able to delegate authority to a stand-in only for the time of the leave but the decisions made by the stand-in should stay valid longer than that. Unfortunately, the proxy can create valid-looking certificates even after losing its authority. He simply writes false dates on them so that they appear to be signed at the time when he was authorized. The problem can be solved with the help of a trusted time-stamping service if such a centralized authority exists. Often it is easiest to let the certificates expire when the mandate of the proxy does and have the master entity revalidate them.

The above limitations are due to fundamental properties of the access control mechanism. There are, however, some other respects in which we have deliberately satisfied us with less than the maximal expressive power. For example, symbolic expressions could be allowed in conditional certificates. The extension would make it possible to express general rules while the certificates proposed in this paper can only speak of fixed keys. The reason for presenting the less general model here is that it solves the practical problems with the basic delegation certificates that we have met in applications. Future work on symbolic conditions will determine the extent to which they are worth the increased complexity.

The question of the optimal expressive power for the certificates involves issues of computational and communication complexity and typical usage patterns in applications.

A question that we will leave open is the exact structure of the authorizations. The types of access rights and the policies for combining them depend on the application. In PolicyMaker [8,9], the authorizations are expressed as small programs of a safe programming language and certificates can communicate with each other. This maximally generic approach leads to concerns about the tractability of access control decisions [10]. But no matter how the authorizations are encoded, certificates have one general limitation in this respect: they can effectively express qualitative authorizations but not quantitative. That is the topic of the next section.

4.2 Accounting and Redelegation

Accounting of service usage is an essential function in many commercial applications. The delegation certificates themselves do not support accounting in any way. They are inherently reusable and can be combined with any number of access requests without losing their validity. For example, there is no use-once certificate or quota on the number of times a service can be used. Such features can, of course, be built with additional infrastructure but they cannot be encoded into certificates.

If accounting is difficult, a natural alternative is to charge a flat rate per client. But as we argued in Sec. 2.3, it is an equal challenge to keep the subjects from sharing their rights with others. By sharing their access rights, the clients can frustrate the flat rate charging policy.

The reason behind these problems is again the lack of unique identities for the individuals in the system and the lack of a global authority that could police the actions of the individuals. There are three main approaches for overcoming the problems with accounting and charging:

1. Make the charging recursive. Every reseller will be responsible for collecting payments from the clients it delegated rights to or for dividing quotas between them. Apart from physical control of the clients, there are two ways in which the reseller can divide the services and the costs between its clients.
   (a) The reseller divides the service capacity at its disposal into smaller time slots and more specific methods of access. Because the authorizations are refined to be suitable only for very narrow purposes, the clients must repeatedly request new certificates from the reseller who collects usage data.
   (b) The authorized clients may use the services at any time and the server collects usage data. The usage statistics and the certificate chains that were used as proof of access rights are propagated from the server down the tree of resellers.
2. Require the client keys in the system to be certified by a trusted entity that guarantees their payments or gives them a credit rating. The server verifies

the credit before allowing access and collects payments directly from the clients. This may not stop the sharing of access rights but it means there is someone to pay for the metered usage.

3. Require the participants to incorporate a tamper-resistant police module on their systems. Only keys on the tamper-resistant modules are allowed to participate in the distribution of the access rights. The module can do the accounting or enforce whatever limitations are wanted.

All these techniques incur a cost in that they require additional infrastructure and make parts of the system less independent. But these costs are inherent to any accounting and charging mechanism. With delegation certificates, we can decide separately for each application if such measures are needed and if their cost is acceptable.

In Calypso, we have chosen the approach 1(b) because the delegation chains between SPs are relatively short (often only one step) and the charging for component services is arranged in the same tree-like manner. Access rights and payments flow in the opposite directions in the service composition tree (Fig. 4).

A problem related to accounting is access rights *transfer* where an entity giving rights to others loses them itself. From a chain of certificates, it is not possible to see if the chain ends there or if the rights have been redelegated further. Thus, an entity that has redelegated its rights can still use them. Implementing transfer in a distributed system requires a TCB or tamper-resistant police modules. (The tamper-resistant module can, in fact, be thought of as a TCB.) One such system for the transfer of software licenses between tamper-resistant smart cards is described in [5].

4.3 Certificate Revocation

Sometimes an entity distributing access rights may want to reverse its decision after the rights have already been granted. The change in mind may be due to changed circumstances or more accurate information about the subject. In a certificate-based architecture, this means invalidating certificates after they have been issued but before their expiration dates. In general, any decrease in the trust placed on the subject may require the issuer to sign and new certificate and to cancel the old one.

In systems where all access is controlled by one reference monitor, access-right revocation is a simple matter. It suffices to update the access control lists or to store information on the exceptions at the place where the rights are verified.

In a distributed system, ACLs are stored and decisions to grant access are made in more than one place. It is necessary to propagate the information on revoked access rights to all these places. The communication causes unpredictable delays and, consequently, real-time revocation cannot be achieved like in a centralized system. An efficient infrastructure for propagating the revocation information is a central part of many access control systems (for example [17,21]). Some options are to broadcast revocation events or to notify only the interested

parties, to immediately propagate every single revocation or to periodically update exception lists, and to transfer the information in push or pull fashion.

All the revocation methods add complexity to the access control architecture. They imply frequent communication and on-line presence of servers and clients. On open networks, it would be unreasonable to require all entities to set up the infrastructure for efficient revocation. Therefore, we prefer to avoid revocation as far as possible. Instead, we suggest limiting the validity time and authorization of a delegation certificate. They should reflect the level at which the subject can be trusted not to misuse the rights. If revocation is needed frequently, it may be just as feasible to require frequent refreshing of the certificates. The TAOS [25] operating system is an example of an architecture that relies completely on expiration dates instead of revocation.

If revocation is nevertheless needed, the implementation should follow two principles. First, the cost of the additional infrastructure should be paid only by the entities that want the service and only when it is strictly needed. If the issuer of a certificate wants to be able to revoke it, he should be responsible for setting up the distribution channels for the notification. This may be a simple task if the certificates for the rights in question are always verified in a fixed set of servers and extremely complex if there are unregistered off-line verifiers. Second, no system should completely abandon expiration times in favor of revocation. If expired certificates are not purged from the revocation lists, the lists can grow uncontrollably. The more frequently revocation occurs, the shorter the validity periods should be. Finding the right balance that minimizes the costs requires careful analysis of each individual system.

SPKI certificates can include instructions for downloading a lists of revoked certificates or for asking confirmation from an on-line server. If the issuer of a certificate wants to be able to revoke it, he must explicitly state this in the certificate. This way the cost of the revocation lists is paid only when it is strictly needed. Moreover, there is no centralized system for maintaining and distributing the lists. Instead, each certificate can have a reference to the particular server and the signature key that provide the lists.

With conditional certificates, the confirmation from an on-line server can be implemented in the same way as the quality check in Fig. 10. Normally the validity period of the quality certificates from the QC is long or infinite. Making it very short would be the same as asking the SP to get a frequent on-line confirmation from the QC.

4.4 Auditing and Anonymity

Designers of secure access control have traditionally emphasized audit capabilities. That is, every action should be traceable to an entity that can be held responsible for it. The shift from closed military networks to open commercial ones has brought a conflicting need, privacy. Although auditability and anonymity are both desirable security goals, it is not easy to achieve both simultaneously. Hence, we must try to balance the requirements according to the application.

The signature key does not directly reveal who is responsible for a signed request which makes it difficult to trace the responsible parties. For auditing, the keys must be bound to the persons or legal entities that are liable for their actions. That kind of bindings can be created by identity-escrow agents that guarantee to find a responsible person if the need should arise. The escrow agents issue certificates to the keys whose identities they have escrowed. The services that require auditing only accept clients with the escrow certificates.

The key-oriented system protects the users' privacy by not explicitly revealing their names. However, the keys are easily recognizable identifiers that can be used to combine data collected from different sources. Therefore, further measures are needed for reliable privacy protection. The certificate reduction (see Sec. 2.2) helps in some cases. A chain of certificates may reveal the identities of the intermediate entities but when the chain is reduced, that information is hidden. SPKI puts great emphasis on privacy aspects and relies mostly on the reduction.

An alternative anonymity technique is to create temporary keys that do not reveal their owner. When a subject entity wants its anonymity protected, it provides the issuer of a new certificate with a freshly generated public key. The temporary keys cannot be recognized and connected to the owner or to each other. This is often preferable to certificate reduction because the entity responsible for generating the temporary keys is the one whose anonymity is at risk. Although the generation of the temporary keys is costly, it can be done off-line in advance. With both techniques, however, the cost of privacy is an increase in communication and synchronization between entities.

5 Conclusion

This paper described delegation certificates and some of their applications in distributed access control. The goal was an abstract understanding of the basic ideas without implementation details. We found that the main advantages of the certificates lie in decentralization. We also introduced conditional certificates that help in further distribution of management operations in the system.

Delegation catches well the spirit of what is natural to access control of distributed digital services. Some access control policies require additional infrastructure such as a TCB or trusted servers. We feel that such costs should be avoided wherever possible. When their limitations are kept in mind, the delegation certificates can satisfy many every-day access control needs and can be used as a uniform basis for distributed discretionary access control.

Acknowledgements

The work was funded by Helsinki Graduate School for Computer Science and Engineering (HeCSE) and Academy of Finland. In am thankful to professor Olli Martikainen and to Petteri Koponen and Juhana Räsänen for allowing the use of Calypso as a case study. Part of the work was done while the author was at UC Davis Computer Security Laboratory.

References

1. Martín Abadi. On SDSI's linked local name spaces. In *Proc. 10th IEEE Computer Security Foundations Workshop*, pages 98–108, Rockport, MA, June 1997. IEEE Computer Society Press.
2. Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.
3. Tuomas Aura. Fast access control decisions from delegation certificate databases. In *Proc. 3rd Australasian Conference on Information Security and Privacy ACISP '98*, volume 1438 of *LNCS*, pages 284–295, Brisbane, Australia, July 1998. Springer Verlag.
4. Tuomas Aura. On the structure of delegation networks. In *Proc. 11th IEEE Computer Security Foundations Workshop*, pages 14–26, Rockport, MA, June 1998. IEEE Computer Society Press.
5. Tuomas Aura and Dieter Gollmann. Software license management with smart cards. In *Proc. USENIX Workshop on Smartcard Technology*, Chicago, May 1999. USENIX Association.
6. Tuomas Aura, Petteri Koponen, and Juhana Räsänen. Delegation-based access control for intelligent network services. In *Proc. ECOOP Workshop on Distributed Object Security*, Brussels, Belgium, July 1998.
7. D. Elliott Bell and Leonard. J. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, The Mitre Corporation, Bedford MA, USA, March 1976.
8. Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The role of trust management in distributed systems security. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, LNCS. Springer, 1999.
9. Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proc. 1996 IEEE Symposium on Security and Privacy*, pages 164–173, Oakland, CA, May 1996. IEEE Computer Society Press.
10. Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance checking in the PolicyMaker trust management system. In *Proc. Financial Cryptography 98*, volume 1465 of *LNCS*, pages 254–271, Anguilla, February 1998. Springer.
11. David F. Brewer and Michael J. Nash. The Chinese wall security policy. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 206–214, Oakland, CA, May 1989. IEEE Computer Society Press.
12. *Recommendation X.509, The Directory - Authentication Framework*, volume VIII of *CCITT Blue Book*, pages 48–81. CCITT, 1988.
13. Carl M. Ellison. Establishing identity without certification authorities. In *Proc. 6th USENIX Security Symposium*, pages 67–76, San Jose, CA, July 1996. USENIX Association.
14. Carl M. Ellison, Bill Franz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylönen. SPKI certificate theory, Simple public key certificate, SPKI examples. Internet draft, IETF SPKI Working Group, November 1997.
15. Carl M. Ellison, Bill Franz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylönen. SPKI certificate theory. Internet draft, IETF SPKI Working Group, October 1998.
16. M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson. The digital distributed system security architecture. In *Proc. National computer security conference*, pages 305–319, Baltimore, MD, USA, October 1989.

17. Li Gong. A secure identity-based capability system. In *Proc. 1989 IEEE Symposium on Research in Security and Privacy*, pages 56–63, Oakland, CA, May 1989. IEEE, IEEE Computer Society Press.
18. J. Kohl and C. Neuman. The Kerberos network authentication service (V5). RFC 1510, IETF Network Working Group, September 1993.
19. Petteri Koponen, Juhana Räsänen, and Olli Martikainen. Calypso service architecture for broadband networks. In *Proc. IFIP TC6 WG6.7 International Conference on Intelligent Networks and Intelligence in Networks*. Chapman & Hall, September 1997.
20. Ilari Lehti and Pekka Nikander. Certifying trust. In *Proc. First International Workshop on Practice and Theory in Public Key Cryptography PKC'98*, volume 1431 of *LNCS*, Yokohama, Japan, February 1998. Springer.
21. Nataraj Nagaratnam and Doug Lea. Secure delegation for distributed object environments. In *Proc. 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, pages 101–115, Santa Fe, NM, April 1998. USENIX Association.
22. A guide to understanding discretionary access control in trusted systems. Technical Report NCSC-TG-003 version-1, National Computer Security Center, September 1987.
23. Pekka Nikander and Lea Viljanen. Storing and retrieving Internet certificates. In *Proc. 3rd Nordic Workshop on Secure IT Systems NORDSEC'98*, Trondheim, Norway, November 1998.
24. Ronald L. Rivest and Butler Lampson. SDSI — A simple distributed security infrastucture. Technical report, April 1996.
25. Edward P. Wobber, Martín Abadi, Michael Burrows, and Butler Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32, February 1994.
26. Philip Zimmermann. *The Official PGP User's Guide*. MIT Press, June 1995.