YKI KORTESNIEMI

MANAGING THE USAGE OF AUTHORISATION CERTIFICATES

LICENTIATE'S THESIS

May 30, 2003

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

HELSINKI UNIVERSITY OF TECHNOLOGY
Deparment on Computer Science and Engineering

ABSTRACT OF THE LICENTIATE'S THESIS

Author
Yki Kortesniemi

Date 30.5.2003
Pages 41+42

Title of the Thesis
Managing the Usage of Authorisation Certificates

Language
Englanti

Chair
Formaalit menetelmät tietojenkäsittelytekniikassa

Chair code
T-79

Supervisor
Professori Hannu Kari

Instructor
Professori Hannu Kari

Limited valuable resources need protection from unintended users and excessive usage. This problem can be solved using access control of some form. Many good technologies exist for centralised systems, but distributed systems present interesting challenges as the technologies are not ideally suited for situations like multiple alternative resources, distributed management or anonymous users.

A proposed solution, SPKI authorisation certificates, naturally provide many of the required characteristics, but they are inadequate to protect limited resources against exploitation. They cannot support use cases where the right can be used e.g. only a certain number of times or up to a specified amount. Instead, they always grant unlimited access.

In this thesis, the author analyses the SPKI certificate model, identifies the missing elements and provides the necessary additions. The resulting model enables numerous new application areas. The model is then analysed from points of view of usability, security and scalability. The author concludes that good usability is achievable with careful design, that the new model has no new substantial security weaknesses, but that the issue of scalability still merits further work.

Keywords: authorisation certificate, SPKI, validity management, revocation

TEKNILLINEN KORKEAKOULU **LISENSIAATINTUTKIMUKSEN**
Tietotekniikan osasto **TIIVISTELMÄ**

Tekijä Yki Kortesniemi	Päiväys 30.5.2003 Sivumäärä 41+42
Työn nimi Managing the Usage of Authorisation Certificates	Kieli Englanti
Professuuri Formaalit menetelmät tietojenkäsittelytekniikassa	Professuurin koodi T-79
Työn valvoja Professori Hannu Kari	
Työn ohjaaja Professori Hannu Kari	

Rajallisia arvokkaita resursseja tulee suojata asiattomilta käyttäjiltä ja liialliselta käytöltä. Tämä ongelma voidaan ratkaista sopivalla pääsynhallintajärjestelmällä. Keskitetyille järjestelmille on olemassa monia hyviä ratkaisuja, mutta hajautetut järjestelmät tuottavat haasteita koska ratkaisut eivät luontevasti sovellu sellaisiin tilanteisiin, joissa on lukuisia vaihtoehtoisia resursseja, joissa tarvitaan hajautettua hallintaa tai joissa käyttäjien anonymiteetti halutaan turvata.

Eräs ratkaisuehdotus, SPKI-valtuussertifikaatit, luonnostaan mahdollistaa monet tavoitellut ominaisuudet, mutta ne ovat riittämättömiä suojaamaan rajallisia resursseja liikakäytöltä. Ne eivät sovellu käyttötapauksiin joissa oikeutta voi käyttää esimerkiksi vain tietyn määrän kertoja tai vain määriteltyyn summaan asti, koska ne antavat aina rajattoman käyttöoikeuden.

Tässä työssä kirjoittaja analysoi SPKI-sertifikaattimallia, tunnistaa siitä puuttuvat osat ja tarjoaa tarvittavat täydennykset. Laajennettu malli mahdollistaa lukuisia uusia sovellusalueita. Seuraavaksi mallia analysoidaan käytettävyyden, turvallisuuden ja skaalautuvuuden näkökulmasta. Malli mahdollistaa hyvän käytettävyyden, jos suunnittelu tehdään huolella, ei sisällä isoja uusia turvaonglmia, mutta skaalautuvuus vaatii vielä lisätyötä.

Avainsanat: valtuussertifikaatit, SPKI, voimassaolon hallinta, revokaatio

Summary of Contents

Table of Contents

ACKNOWLEDGEMENTS

First, I would like to thank Dr. Pekka Nikander and Dr. Arto Karila for getting me started in the field of security and setting some high goals. The numerous discussions I have had with Pekka during these years have further helped me advance my thinking.

This thesis is the result of working in the TeSSA research project at Helsinki University of Technology and at the STAMI research project at Helsinki Institute for Information Technology. I am grateful to all my co-workers in these projects for their contributions to my thinking. I am especially indebted to my co-authors Tero Hasu, Jonna Särs, Kristiina Karvonen and Antti Latva-Koivisto for their participation in two of the publications that form part of this thesis.

I am particularly grateful to my instructor and supervisor Prof. Hannu Kari for the guidance and encouragement he has given me over these years.

I would also like to thank the people who helped in the writing of the introductory part of thesis: Dr. Martti Mäntylä for his helpful advice on structuring and focusing the work, Ms. Katja Kuusikumpu for the numerous suggestions that helped clarify the presentation, and Mr. Benjamin Vary who once again helped me polish the language.

Finally, I would like to thank Antti for his understanding during this process.

Espoo, May 30th 2003

LIST OF PUBLICATIONS

This thesis consists of two parts. The first part provides a concise introduction to the problem area, describes the main results, analyses their significance and provides pointers for future work.

The second part is a collection of articles related to the theme of the thesis published during the years 2000-2002. The publications are:

> [Publication I] Yki Kortesniemi, Tero Hasu, Jonna Särs: A Revocation, Validation and Authentication Protocol for SPKI Based Delegation Systems, Proceedings of Network and Distributed System Security Symposium (NDSS 2000), 2-4 February 2000, San Diego, California.

This paper discusses the requirements for authorisation certificate revocation, introduces two new revocation methods for SPKI and defines a protocol for verifying SPKI certificate validity at the time of usage. The author was responsible for the initial idea behind this paper, but the final result was the product of tight co-operation with the participating writers, each providing roughly equal input.

> [Publication II] Kristiina Karvonen, Yki Kortesniemi, Antti Latva-Koivisto: Evaluating Revocation Management in SPKI from a User's Point of View, Proceedings of Human Factors in Telecommunication 2001, Bergen, Norway, 2001.

This paper looks at the SPKI revocation methods from the user's point of view and gives system designers advice on how to implement usable systems. The author is responsible for the initial idea behind the paper and its technical content.

[Publication III] Yki Kortesniemi: Validity Management in SPKI. Proceedings of the 1st Annual PKI Research Workshop, Dartmouth College, Hanover, New Hampshire, USA, April 2002.

This paper presents a protocol for managing the validity of SPKI certificates.

[Publication IV] Yki Kortesniemi: SPKI Performance and Certificate Chain Reduction. Informatik 2002, Workshop "Credential-basierte Zugriffskontrolle in offenen, interoperablen IT-Systemen", Dortmund, 30.9.- 3.10.2002.

This paper discusses the problems resulting from the need to evaluate long certificate chains.

1. INTRODUCTION

Valuable resources, such as our home or credit account, require protection so that only the intended people can indeed access them. To achieve this, various access control solutions are used. For instance, the lock on our front door prevents unwanted guests from entering our home and thus, provides us with privacy. Also when shopping, our credit account can be used only with the corresponding credit card. But the need for control is not limited to private property - one example is the public transport system: without control, a number of passengers would probably forgo buying a ticket. To avoid the freeloaders, all valuable resources need some form access control.

Physical solutions, like a key for instance, unfortunately have some significant shortcomings: if we lose our front door key, our only recourse is to have the lock changed and to issue new keys to all family members, because we cannot just revoke the lost key. If instead we are talking about the door to a large company, the cost of replacing all the locks and keys obviously becomes a much bigger problem. The solution to this kind of situation has been to use digital technology for keys. It then becomes possible to revoke individual keys without having access to the physical token containing the key.

Digital solutions also have the capability to operate over networks. Paper bills cannot be used to pay purchases on the Internet, but a credit card is up to the task. Alas, credit cards are not secure enough. A credit card was a good choice for on-the-place purchases, but the situation is completely different when shopping over the Internet, where it is impossible to verify the possession of the card. Then, the right to pur-

chase is granted solely based on the knowledge of the contents of the card, which makes it too easy to just copy this information and misuse it.

There already exist good digital access control solutions for centralised systems, which also work on the network. However when we start to talk about larger, distributed systems we discover that the centralised access control solution is holding us back. We need a solution that works effectively in systems with large user-bases, multiple resources and distributed management.

In this thesis, I examine this problem in detail and identify a more promising approach, which I then extend so that it can be used to solve many everyday usage scenarios that it previously could not solve. The resulting solution has advantages over previous solutions, particularly in large distributed systems.

The rest of the thesis is organised as follows: Chapter 2 gives the necessary background on access control. Then, Chapter 3 defines the problem and criteria for evaluating the solution and introduces the example cases that the solution must be able to solve. Chapter 4 describes the solution developed and Chapter 5 analyses and evaluates the results. Chapter 6 proposes ideas for future work and finally, Chapter 7 presents my conclusions.

2. BACKGROUND

In this chapter, we shall first glance at the concept of access control, then we shall look at the main approaches of digital access control technologies and finally, focus on one of them, capability and certificate based access control, with special regard to the Simple Public Key Infrastructure (SPKI) authorisation certificates.

2.1. PHASES OF ACCESS CONTROL

The access control process can be said to consist of the following phases (depicted in Figure 1)[Publication I]:

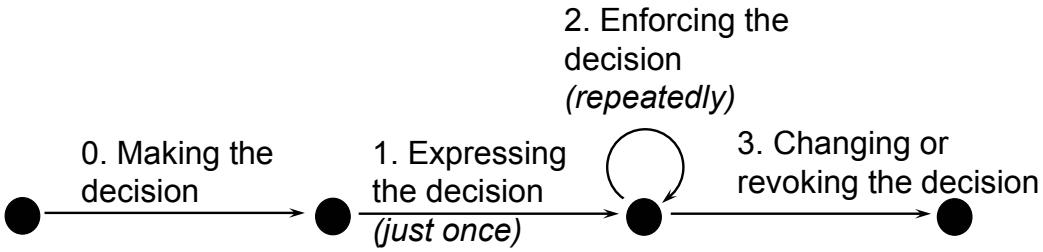

Figure 1: Phases of access control.

In Phase 0, someone either owning the resource or having the right to control access to it, known as the *issuer*, makes the decision to grant a user, known as the *subject*, the permission, known as the *right*, to utilise the resource within set limitations. This decision could be based on things like the issuer knowing the subject (a friend), the subject holding some position in the issuer's organisation or the subject being a pay-

ing customer to the issuer's service. In our example, the issuer might be the sales office of a transport service and the subject is a passenger who, wants a ticket, is willing to pay for it and in fact, has initiated the whole phase by entering the sales office. Hence, we notice that Phase 0 can be initiated by either the issuer or the subject.

Next, in Phase 1, the issuer must somehow express the decision in a form that can later be used to verify the subject's right to use the resource. For instance, the passenger could be issued a ticket, which the passenger then shows whenever she wants to use the resource, i.e. travel. Naturally, this ticket has to be such that the passenger is unable to manufacture tickets herself or modify a valid ticket to grant her additional rights, for instance, extend the validity of the ticket.

In Phase 2, whenever the subject tries to use the resource, the *verifier* (also known as the *validator*) enforcing the access control at the resource makes sure that the right still exists. In our example, this could be the driver who inspects the passenger's ticket when she steps into the bus. This validation process entails checking the subject's right to the operation she attempts to perform, i.e. that the ticket is valid for the intended trip and not, for instance, for a trip in another zone. The process also verifies that the subject is indeed the correct user of the right, i.e. that the passenger is not using someone else's ticket. In practice, having a human verifier is feasible only in a very limited set of cases and most often an automated solution is preferable, because it is less prone to errors and tends to be more economical, particularly in applications with a large user-base. Naturally, for the validation process to be automated, the rights created in Phase 1 have to be expressed in a machine-readable format.

Compared to Phase 1, which only takes place once, Phase 2 can be repeated numerous times. Therefore, it makes sense to design the access control solution so that Phase 2 is as simple to perform as possible, even at the expense of Phase 1.

Should the passenger lose the ticket, the issuer might be able to *revoke* the ticket (Phase 3) and issue a replacement. And if the certificate does not wind up revoked, it will eventually expire when the subject exhausts the right or its validity period simply runs out (Phase 4). Therefore, both Phases 3 and 4 bring the access control life cycle to an end.

2.2. DIGITAL ACCESS CONTROL

Access control can be implemented in many ways, for instance a physical key can be used to control access through our front door as we previously discussed. Physical solutions, unfortunately, cannot be used over the network, a realm reserved expressly for digital solutions. As the focus of this thesis is to find a solution that works also over the network, we shall concentrate solely on digital access control.

Historically speaking, digital access control (hereafter referred to as: *access control*) started in the form of an *access matrix*, which listed all authorized entities and their rights in a table. As the number of users and the possible rights they could have grew, the table ended up growing very large – and yet, it was mostly empty, as each user only had a small subset of all possible rights. The solution was to split the table giving us two very different options: access control lists and capabilities.[1]

Traditionally the popular choice has been to base access control solutions on the concept of *Access Control List* (*ACL*), where every resource is bundled with a list of authorised users. Typically, the list is located next to the resource, with all the relevant information in one place. An example could be the VIP list at the door of a club. In this solution, when the issuer wants to create a new right or change an existing one, she merely has to change the list. Furthermore, because the list is in the issuer's control, it is relatively easy to protect the *integrity* of the list, i.e. to make sure that the subject or any other outsider cannot change the contents of the list once it has been created and delivered to the door, and thus create new or extended rights. Moreover, because only the issuer is able to modify and issue lists, we can be sure that all information is *authentic*, i.e. it comes from the correct, stated source.

The downside of this solution is that the issuer cannot make any changes if she cannot access the list. So, if the club manager is enjoying her vacation on a long hike without any means of communication, she is *offline* and cannot make any changes to the list until she becomes *online* again. In addition, if there are several doors to the club, we need a connection from all the doors to a central list, but guaranteeing an always-available connection is not easy. Alternatively, we could have several copies of the list, but keeping them up to date requires extra work. Moreover, what if we

have several clubs around the country all accepting the same VIPs and several club directors granting the VIP status to new people – all the managers have to be able to connect to the list.

Capabilities reverse the concept, turning the centralised system into a distributed one. In a capability-based system, the users of the resource are given a ticket that proves they have the right to use the system. The right no longer resides with the resource but with the user (all the VIPs have a special card they show at the door) and the right automatically follows the users to whichever copy of the resource they go. Therefore, if the club has several entrances or there are several clubs around the globe, the VIP can use any one of them. Also, new capabilities can be created and given to the subject without any connection to the resources. Hence, if the manager meets a new VIP while hiking, she can create a new VIP card with which the VIP can then go clubbing while the manager continues her hike. So, we notice that in large distributed systems, the capability-based approach has some inherent advantages. Therefore, with our network focus, we shall concentrate on the capability-based approach for the purposes of this study.

2.3. DIGITAL SIGNATURES AND PKI

With regard to capabilities, two big problems are their authenticity and integrity: how do we know that they come from the stated source (e.g. the club manager) and how do we know that they have not been tampered with? Luckily, both of these problems can be solved with *digital signatures*.

Digital signatures are a product of Public Key Cryptography (PKC). PKC is based on the idea of using two keys: one private, which is known only to its owner, and one

public, which can and should be known by the rest of the world. With the private key, the owner can *sign* any digital document thereby creating a digital signature.[1]

A digital signature has the following interesting properties:

- It cannot be created without using the private key, thus preventing anybody else from creating fake signatures.
- It is unique to that document: it is not possible to create or find any other document to which that signature would apply.
- It can be verified with the corresponding public key: as long as we know the owner of the public key, we also know who created the signature.

By making sure that the intended recipients know the owner of a public/private key pair, the owner can create signatures and convince the recipients of their authenticity as well as the integrity of the document. Therefore, to protect the VIP capability, the club owner merely has to sign them (a signed capability is also called a *credential*).

The verifiers at club doors have to have the club owner's public key to verify the signatures. Moreover, it has to be the correct key: if a malicious outsider manages to introduce his own public key as the club owner's key, the outsider can then pretend to be the owner and create new VIP credentials.

The problem of securely distributing the correct public keys has been solved with *Public Key Infrastructures* (*PKIs*), where trusted parties make statements about who owns each public key. Later, the term PKI has been expanded to mean systems where, in addition to names, other attributes such as rights can be bound to public keys, as we shall see in the next section. The tool used to carry the statements in a PKI is known as a *digital certificate*.

---

[1] Actually, the signature is created by using the private key on the *secure hash* (hereafter referred to as: hash) of the document. A hash is a short, fixed length number identifying that document – all documents regardless of their length have unique hash values (theoretically, there is only a limited number of hashes and hence, only a limited number of documents can have a unique hash, but as this number is very large, in practice we can say that all documents have a unique hash).

2.4. DIFFERENT CERTIFICATE TYPES

A digital certificate (hereafter referred to as: *certificate*) is a fixed form document, where a signature is used to guarantee the integrity of the information within. In practice, a certificate is used to bind two out of three possible things together and to tell something additional about that relationship. Hence there exist three major types of certificates: identity certificates (e.g. X.509[10] and PGP[2]), authorisation certificates (e.g. SPKI[8]) and attribute certificates (e.g. extension in X.509) as shown in Figure 2 [Publication III].

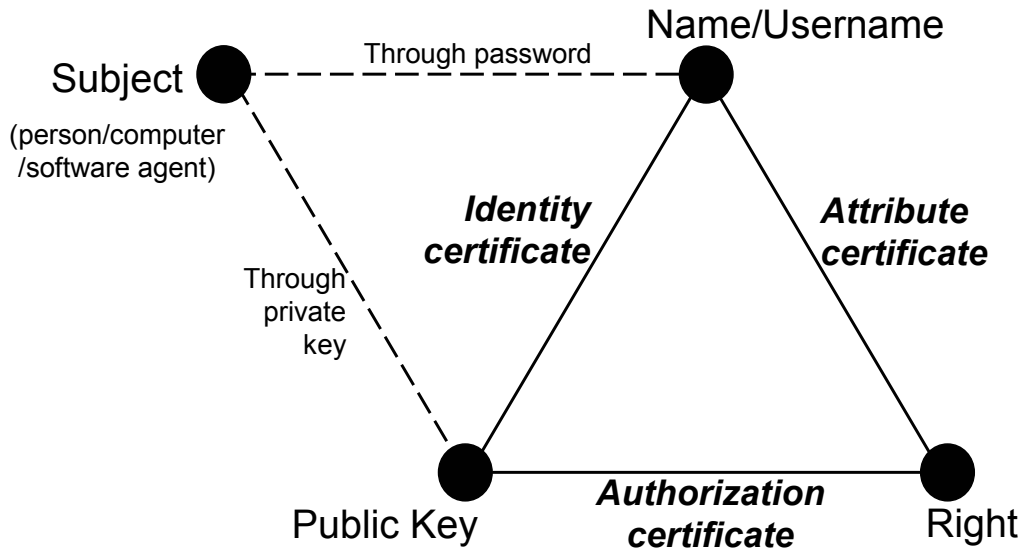

Figure 2. Three major types of certificates.

In an *identity certificate*, a trusted third party testifies his belief that a particular key (a public key - certificates only contain public keys as the private key has to be kept secret) belongs to the subject indicated by *name* (this can be e.g. the person's real name or a username). Of course, to be usable, this entails that the issuer and thus, the signer of this identity certificate (typically an organization called Certification Authority, CA) actually makes sure that the key is controlled by the said entity. Also, as the CAs are the only parties issuing certificates, and as all access decisions are based on these, all users of the system will have to trust the CAs.

An *authorisation certificate*, on the other hand, binds a right (a right is sometimes called an *authorisation*) to the subject's public key (sometimes also to the hash of an object). Furthermore, because it is bound to the public key of the subject, the possession of this certificate alone will not grant any rights; anyone wishing to use the certificate also has to prove the possession of the corresponding private key. Authorisation certificates can be issued by anyone owning a resource or having the right to grant access to someone else's resource. This means that potentially every human, computer, or even a software agent could issue certificates. This difference in the number and resources of issuers between the two certificate types has significant implications on the revocation systems used, as we shall later discuss.

The third and less common type, the *attribute certificate*, is used to bind an authorisation to a name – the same binding that an ACL does. For the purposes of this thesis, we will not discuss attribute certificates any further, but shall instead concentrate on the first two types, authorisation certificates in particular.

An important feature of authorisation certificates is that they can be used to delegate the rights they carry unless it is expressly forbidden in the certificate. The subject can delegate the right or part of it to someone else without any help from the issuer - a feature, which makes distributed management easier to organise than in centralised solutions. For instance, we can implement a scheme, where a parent can issue a copy of her credit card to a child while still keeping her own credit card [2].

2.5. USING CERTIFICATE FOR ACCESS CONTROL

To better appreciate the differences between identity and authorisation certificates, let us briefly look at how they are utilised [Publication III]. In Phase 0 of the access control process as shown in Figure 1, certificates play no role, so we shall next look at Phases 1 and 2 (Enforcing the decision).

For the right to be usable to the subject, we need to establish in Phase 1 a binding between the subject requesting access and the required right. As we can see from Figure 2, there are several ways of doing this. In all of these, the binding between the

subject and the key is assumed much tighter than the binding between subject and username. The latter binding is protected with a password, which in practice has been proven much less secure, so from here on, we shall use the binding to the key. This tight binding with keys however, does not always hold, as the subject can either lose the control or just give the required private key away. In both these situations, revocation of that key and the associated rights is normally required.

We start by acquiring the subject's public key, which can be accomplished e.g. either with an identity certificate (if we know the subject's name) or from the subject directly in a face-to-face meeting, in which case it is not always necessary to even know the subject's name.

Once we have the key, we have essentially two routes to reach the right: we can use an identity certificate to bind the key to a name and then create either an ACL entry or an attribute certificate binding that name to the right, or we can bind the key directly to the right with an authorisation certificate.

This first approach nicely extends existing solutions, which usually are based on ACLs, but it also has its problems:

- By design, it makes anonymous usage impossible. In some systems, it is a requirement to prevent anonymous usage, but in other cases it merely promotes unnecessary monitoring of users.
- Making a tight binding through the name is not easy, as it requires names that are unique within the application domain – otherwise namesakes can share their rights. If we have a small organisation this might be quite feasible, but if we aim for global consumer applications, we need globally unique names which are difficult for humans and impractical for computers.

The final problem affects Phase 2 in which we have to (repeatedly) prove the existence of a binding from the subject to the right:

- The binding from a key to an authorisation is unnecessarily long – it consists of two steps: key to name and name to authorisation. This is an important

aspect, as the verification of this binding will be performed many times – in fact, every time the subject uses the resource.

An authorisation certificate, on the other hand, makes a direct binding from the key to the authorisation. This makes the binding simpler, but also anonymous. In reality, the key is not totally anonymous but an alias or a pseudonym, but since these pseudonyms do not have to be registered anywhere, it can be very difficult to trace them back to the user's identity. This kind of situation could present itself e.g., if an authorisation certificate is used to implement a single-trip bus ticket which the subject pays for in advance – then, the issuer would have no need to verify the subjects identity. If however the anonymity becomes a problem, it can be circumvented by verifying the subject's identity already in Phase 1 (but should this be omitted, we cannot perform it retroactively).

Because authorisation certificates enable delegation and hence, distributed access management which ACLs cannot accomplish without much complication, and because authorisation certificates provide a more direct, anonymous and more secure binding to the right, we can conclude that authorisation certificates offer a simpler solution for distributed systems than solutions based on identity certificates.

2.6. MANAGING CERTIFICATE VALIDITY

So far we have looked at how certificates are used to grant rights to a subject. These rights however, are seldom limitless. They usually expire after a specified period (Phase 4) and the usage can also be limited in other ways, for instance, by limiting the times a certificate can be used. Also, they sometimes even have to be revoked before they would naturally expire (Phase 3). There are essentially two reasons to revoke a certificate:

- Issuer discovers or wants to prevent misuse (=Issuer initiated revocation).
- Subject loses control of the certificate and wants a replacement issued (=Subject initiated revocation).

Regardless of the initiator, it is always the issuer that actually revokes the certificate. Collectively, the methods used to limit certificate usage and to revoke them are called *validity management*.

The basic method for limiting certificate validity, which most certificate types have in common, is *validity period* dates. They are often called the *not before* date and the *not after* date. Validity periods are easy and efficient to check, even in an offline environment. However, validity periods alone do not always suffice, the need to revoke a certificate may arise long before the certificate was originally planned to become outdated. The longer life span the certificate has, the longer is the potential period during which the certificate is spreading false information, but if certificates with very short validity periods are used to reduce the risk, the management overhead might easily grow too large.

Certificate Revocation Lists (*CRLs*) are the most common revocation method used in combination with validity periods. A CRL is a signed list issued by the certificate issuer identifying all revoked certificates by their serial numbers or some other reliable identification (e.g. a hash). If the certificate is not on the list, it is assumed valid. The list includes a time stamp or a validity period. The CRLs are published on a periodic basis, even if there are no changes, to prevent replaying old CRLs.[12]

The main problem with CRLs is that they only shorten the period of possibly false information being accepted as correct - they do not eliminate it. Furthermore, the verifier has no control over how often the CRL is updated, and thus cannot affect the amount of risk it is accepting [15]. The CRLs also may get very long, requiring a lot of bandwidth, a large storage capacity and excessive processing. There have been several proposals for improving the performance of the CRLs including Delta-CRLs and Certificate Revocation Trees [12]. Their effect on performance varies, but their revocation characteristics are the same as a CRLs.

If all the parties can be assumed to stay online, the most timely way for the verifier to check revocation is to directly ask the issuer or a designated validity server about the certificate in question every time the certificate is used. The issuer or validity server may respond with a simple yes or no together with a timestamp and a signature, or

the reply may also include other information such as a time period during which no further proof of validity is required. Online validation is simple for the verifier, but compared to a CRL, it requires more processing power from the validation server, who must create a signature for each new reply.

Although the online check seems to be very simple, it is flexible enough to allow for a wide variety of validation policies. The validation server could simply say the certificate is valid if it has not been revoked, but it could also keep track of the how many times and how the user has used the certificate, and make the validation decisions based on the context. The different revocation methods have been discussed in more detail in Publication I.

The majority of work done in the field of certificate validity management has so far concentrated on identity certificates, in particular on X.509 identity certificates. Unfortunately, compared to SPKI authorisation certificates, there are a few significant differences in the X.509 model, which prevent us from directly applying all the solutions:

- The number of certificate issuers. In X.509, the number of CAs that issue certificates is orders of magnitude smaller (in SPKI, every human, computer etc. can issue certificates).
- Risk model. In X.509, the issuer and verifier are normally separate entities. The risk is taken by the verifier, yet the revocation decisions are made by the issuer. In SPKI, the risk takers are also issuing the certificates and can therefore control the revocation decisions to balance the risk.

So far authorization certificates have been used in binary ways: either you get the access to the resource or you do not. The resource itself can be defined with very fine granularity, but the result is always the same: if you get the access, there is no limit to how much you can use the resource.[13] With this kind of approach, it is not possible to implement bus tickets, which are good for ten trips, or credit cards with monthly limits.

2.7. THE SPKI CERTIFICATES

The Internet Engineering Task Force (IETF) has been developing SPKI as a more flexible alternative to X.509. SPKI was designed to support certificate-based authorisation but it can also be used to certify identity. However, it should be noted that while X.509 and other name oriented systems use names as a starting point and bind keys to names, SPKI uses cryptographic keys to represent identities and binds rights or names to keys. SPKI has adopted many ideas from the SDSI [16] and PolicyMaker [4] prototype systems. We have concentrated on SPKI because it focuses on the management of rights as opposed to e.g. KeyNote[6], which focuses on mathematical proofs on chain validity.

SPKI authorisation certificates[8] like any authorisation certificates, are signed statements of authorisation. The certificate can be abstracted into a signed quintuple (I,S,D,A,V) where

- *I* is the Issuer's (signer's) public key, or a hash of the public key.
- *S* is the Subject of the certificate, typically a public key, a hash of a public key, a name, or a hash of some object.
- *D* is a Delegation bit.
- *A* is the Authorisation field, describing what access rights the Issuer delegates to the Subject.
- V is a Validation field, describing the conditions (such as a time range) under which the certificate can be considered valid.

The meaning of an SPKI authorisation certificate can be stated as follows: Based on the assumption that *I* has the control over the rights or other information described in *A*, *I* grants *S* the rights/property *A* whenever *V* is true. Furthermore, if *D* is true and *S* is not a hash of an object, *S* may further delegate *A* or any subset of it. The integrity and authenticity of the certificate are protected by a signature created with the issuer's private key. Hence, it is easy to verify the signature with the issuer's public key and be assured that the certificate indeed comes from the issuer.

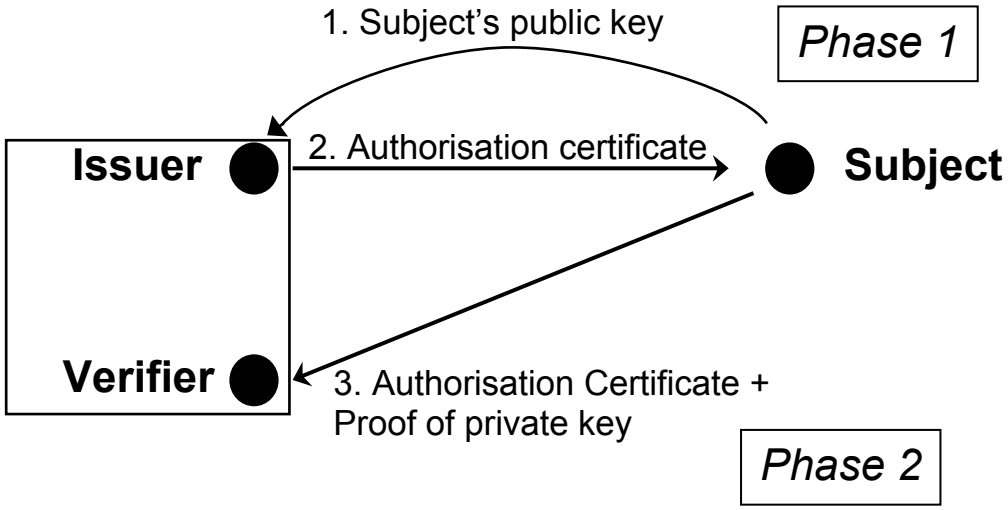

Figure 3: Access control using SPKI certificates

The first two phases of the access control process described earlier are depicted using SPKI certificates in Figure 3. In Phase 1, the subject first provides her public key to the issuer e.g. by handing it over in person. This could be a key already used in some other certificate or a brand new key – in fact, every single certificate could use a different key. Then, the issuer can create the authorisation certificate with appropriate rights and validity conditions. In Phase 2, to use the right, the subject provides the verifier with the certificate and proves the possession of the private key corresponding to the subject's public key in the certificate. The verifier also verifies all the validity conditions and only if they all are valid, the certificate is valid and the right can be used.

As the figure shows, logically the issuer and verifier can be thought of as two different roles of a single entity. In practice, however, usually these roles are divided between two entities: a mechanical verifier that only validates certificates (or chains of certificates) and a more intelligent issuer that makes the decisions to grant the rights. However, if these roles are divided, how does the verifier get the issuers public key so that it can verify the certificate chain? This can be solved by the verifier first granting the issuer a certificate with the right to make all access decisions. All chains now start from the verifier, go through the issuer and end with the subject making the chain validation possible for the verifier.

In SPKI, all the validity management methods are placed in the validation fields. In addition to the validity period, there are three online validity checks: CRLs, *revalidations* (also known as reval) and *one-time* checks. Furthermore, the SPKI theory [7] defines other online checks, but they do not appear in the structure drafts [8] yet.

The validity period definition consists of two parts: not-before and not-after. Both parts are optional, and if either one is missing the certificate is assumed to be valid for all time in that direction. There is an additional type of validity period called *now*, which has a length of 0, and can only be the result of an online check. It is interpreted to mean that the certificate is valid the moment the validation request was made, but it states nothing about the future.

Validating a certificate is relatively straightforward, as all the different validity conditions end up being converted to validity periods. Therefore we only check that the validity period stated in the certificate, as well as the online checks (all replies are validity periods) are all valid at the time of use, and the certificate as a whole is then valid and therefore grants the included permission.

All the online checks are defined in the Validation field of the certificate using the following format (BNF[13] notation is used for the formats):

<online-test>::"(" "online" <online-type> <uris> <principal> <s-part>* ")" ;

Where <online-type> can be crl, reval or one. The <uris> specify one or more Uniform Resource Identifier (URIs)[4] that can be used to request the validity information: e.g. in the case of crl, the URI points to the crl file. <principal> specifies the public key used for verifying the signature on the online reply. The <s-part> is optional, and may contain parameters to be used in the online check. In their replies, all methods identify the certificate they refer to with the hash of that certificate.

CRL is the standard list approach discussed earlier. SPKI includes both traditional and Delta CRLs in its specification. Reval is an online condition, where the response is always valid for some stated period, during which the same reply can be used repeatedly. One-time, on the other hand, is an online condition whose reply is valid only once at the time of requesting - every usage therefore requires a new validation.

3. PROBLEM STATEMENT AND CRITERIA

The goal of this thesis is to improve the SPKI authorization certificate model to support the management of limited resources.

In order to better understand the different limitation levels, let us first classify rights in Table 1 (based on [Publication II]) and then go over use cases for each of them [Publication II]. The classification goes from the widest to the most restrictive, so a higher level is always a subset of the preceding levels.

Table 1: Classification of rights

Type	Name	Description
A	Implicit trust	The right does not expire
B	Expiring right	The right has a validity period
C	Revocable right	Right can be revoked
D	Context dependent	Right is valid only in certain contexts
E	History dependent	Right depends on usage history

Type A is a rare case; it is only applicable to situations like a computer implicitly trusting its administrator to make all decisions.

Type B suits a situation, where the right can be used without limit until it expires and it is not valuable enough to require revocation. An example would be a single use bus ticket valid for one hour.

Type C comes into play when the right is so valuable that revocation capability is required. An example is a bus ticket valid for a whole year – if it is lost, the subject would want it revoked and a replacement for it.

Type D applies to situations, where the context of usage determines whether the right can be used. One example might be a parking building, where a company rents space for its employees. As only a limited number of employees are present at any given moment, a company of 100 employees might rent only 45 parking spaces. Any employee is able to park as long as there are less than 45 other employees already parked in the building.

Type E applies to situations where the right depends on previous usage. An example is a credit card with a monthly limit or a bus ticket with 10 trips.

The problem is therefore to define the necessary changes to SPKI so that it can be used for cases of all types, A-E.

The solution developed should satisfy the following criteria:

- Usability: As users are generally considered the weakest link in security, the usability of the solution should be sufficiently high.
- Security: the new mechanisms should not weaken the security of SPKI access control.
- Scalability: the solutions should be able to scale to global applications.

4. SOLUTION

SPKI has been intended to be a very flexible access control solution. The intended application domains extend from things like organisations which want to control their internal access rights and know their users, to global applications where consumers buy some access rights with cash (e.g. the right to read the current issue of a particular magazine) and want to stay anonymous.

Yet, despite these goals, at their current state SPKI certificates can only be used to solve cases of types A and B. There are validation methods for types C and D, but SPKI lacks the protocols to validate these methods at the time of usage and a protocol for the certificate issuer to change the validity state of a certificate, e.g. to revoke one. Without these, no online method (type C, D and E) can be used. In addition, SPKI lacks a method to implement type E solutions.

Therefore, to solve the problem presented in the previous chapter, the author has designed the following:

- A new method for managing type E situations
- A protocol for validating a certificate at the time of usage
- A protocol for managing the validity of a certificate

4.1. NEW METHODS

Let us start with the new type E method, *limit*. Unfortunately, a certificate alone cannot accomplish history-based validation, as the certificate cannot contain any information about its usage history. The certificate cannot be modified to signify that it

has been used and we cannot take the certificate away from the user and replace it with one that has less right remaining, as the user might have numerous copies of the original certificate – after all, it is good to have backup copies of the certificate and the user might use the same certificate in different devices. Finally, the information about the use cannot be stored separately, as this additional information might "accidentally" get lost, should the user need more credit. The solution chosen was to use an online server that keeps track of the amount of usage. The certificate then contains a reference to an obligatory online check that grants or denies every operation based on the accumulated total. Further, the new method does not limit what kind of things the server can keep track of: they can be trips on a bus or an amount of money spent so far this month or anything else the system designer wants.

The problem in this method is that not everyone can be allowed to perform the validation. With revocation, anyone can be allowed to verify whether a particular certificate has been revoked or not. However, when we talk about the new type E validation, every successful validation also consumes part or all of the right to the limited resource. Therefore, only a party to whom this limited use of resource has been granted either directly or through delegation, can be regarded to have the right to make these validation requests. Otherwise it would be possible for a malicious neighbour to use all of the limit (but not the resource itself) without the rightful parties' consent.

As we discussed earlier, the current SPKI structure includes the validity period (Types A and B) and three online validity checks: CRLs (Type C), revalidations (Type C) and one-time (Type D). So, with the new method limit (Type E), we have all the necessary methods. However, the author defined one additional method to essentially replace the revalidation method: *Renew* offers an alternative approach to revocation. Instead of issuing long-lived certificates and then worrying about their validity, we issue a string of short-lived certificates, which together cover the lifetime of a long-lived certificate. This simplifies matters, as the short-lived certificates can often operate offline and the network connection is required only to automatically fetch the subsequent certificate. If everything is in order, the reply contains the next certificate, but if the right has been cancelled, the reply contains a validity period dur-

ing which renewal requests will be denied (i.e. the conceptual long-lived certificate is not valid during this period). Thus, the benefits of Renew are that it is conceptually simpler and that it also offers the interesting possibility of changing the right and other fields from certificate to certificate.

[Publication I] proposes the formats for the two new online validations: limit and renew.

4.2. CERTIFICATE VALIDATION PROTOCOL

The second issue requiring development is the protocol used to validate the certificate chain at the time of usage – i.e. a protocol to implement Phase 2 from Figure 1. To better understand the problems involved, let us take an example: Helen is in London buying groceries and wants to pay with a credit card that is implemented as an SPKI certificate. In this case, the various parties have the following interests during the transaction [Publication I]:

The merchant is interested in

- receiving the payment for the groceries.

Helen, on the other hand, wants to make sure that

- the payment is indeed received by the merchant and not by an impostor.
- the merchant is not able to charge her more than once.

She could also be interested in

- hiding her identity from the merchant to avoid receiving further publicity material or because she does not want the merchant to keep a record of her purchases.

The credit company (or anyone else, who grants or delegates rights) might be interested in

- limiting her purchases by imposing a monthly limit on our credit card.
- being able to cancel the card, should she misuse it or should the card fall in wrong hands.

A successful validation depends on several things. First, we have to be able to authenticate the participants reliably - otherwise Helen's money might end up with an impostor. So we have to be able to authenticate Helen, the merchant, the verifier and all validation servers. We achieve this using an existing secure transport. The relevant public keys for the validator, validation servers and Helen can be found in the certificate chain: the validator is the originator of the chain, the possible validation server is identified in the validation part of the certificate requiring online validation and Helen is the final subject of the chain. The only new public key is the merchant's key, which Helen has to acquire at the moment of purchase with any of the methods discussed earlier. As parties know each other's public keys and have their own private keys, authentication can be arranged. It should be noted that it is not necessary to authenticate the parties in every transaction type. For instance, while fetching a CRL, it is not necessary to authenticate the validation server as long as the CRL is correctly signed.

Secondly, we have to be able to guarantee that the merchant receives one and only one payment. To accomplish this, Helen would delegate the merchant the right to charge her account by a specified amount and control the number of uses with an online check. This online check could, for instance, be directed to the user's own terminal, which would eventually show that the merchant is requesting to use his right. The terminal could then validate this request once, and later deny any further validation attempts.

The remaining problem is related to the right to make Type E validation requests, which is limited to only those who are able to use the related resource themselves. In practice, this means those entities to whom the limited right was granted, and all other entities to whom this right was further delegated. As the verifier is not a receiver of the right, but rather the originator of the chain, he is not be allowed to make any Type E checks in the chain without explicit permission. Therefore, the user of the resource, i.e. the final receiver in the chain, has to authorise the verifier to validate the certificates by issuing a special *validation certificate* for this particular use of this particular chain. In our example, the merchant would authorise the credit card company to make all the necessary online checks.

After having received the payment certificate from the user, the merchant could contact the credit company, which can validate the certificate chain and, should the chain prove valid, credit the merchant's account. If the chain is not valid, the verifier can notify the merchant, which can then deny delivering the service. The merchant, on the other hand, cannot deny having received the payment, and will therefore be caught should he try to deny the service on the pretence of not being paid.

The traditional SPKI view has been that the validation information is fetched by the *prover* trying to use the certificate (in this case, the merchant or client). However, it may be impossible to equip clients having very limited computing and storage capabilities with the logic needed to acquire certificate chains. One solution would be that the verifier could also take care of completing the chains. The downside is that the verifier could face excessive loads, even denial of service. Another solution would be to introduce third party services for resolving the chains as discussed in Chapter 6.

The SPKI certificate validation protocol is presented in detail in Publication I.

4.3. VALIDITY MANAGEMENT PROTOCOL

The final required element is a protocol to manage validity, so that it is possible to e.g. revoke a certificate. Compared to the validation protocol, the management protocol is simpler, as there are fewer parties involved.

Still, there are a number of issues the protocol has to address. One of the basic things is naming the principal(s) that are allowed to issue revocation commands. The most obvious solution is that the principal, who issued the certificate, is implicitly assumed to have the right to revoke it. However sometimes it would make sense to authorise others to revoke a particular certificate, for instance in a situation, where it is imperative that the certificate is revoked as fast as possible after a breach but the original issuer is not available to perform the revocation.

The issuer might also be interested in following how the certificate is used, particularly if it contains one-time or limit conditions, or if there are several individuals with the ability to revoke the certificate. Therefore, this functionality is included.

Finally, the commands and their replies have to be auditable in case there is dispute over the replies given by the server.

The protocol has been defined with the listed features in Publication III.

5. ANALYSIS

We shall now look at how the developed solution can be used to solve the problem identified in this thesis and how well the solution meets the criteria. We shall start by looking at how the different validation methods work and give some guidelines on their usage. We then proceed to how our cases can be solved within the confines of our criteria and finish by looking at the limitations of our solution.

5.1. SOLVING THE CASES

The problem of this thesis was defined as being able to solve situations of Types A to E using SPKI authorisation certificates. We noted that SPKI already had the necessary methods for most of the cases, but lacked the one required for Type E situations. Such a method was therefore defined, as well as another method of Type C. With these new methods, there are now altogether six different methods to limit the validity of a certificate: validity period, crl, reval, renew, one-time and limit [Publication I]. We have ordered the different methods by increasing capability using the classification presented in Table 2. We can say that the types refer to the validity mechanisms themselves, or to certificates, whose most effective mechanism is of the type mentioned. The relative roles of methods have also been illustrated in Figure 4 [Publication III].

In Chapter 3 we introduced our example cases for each type and [Publication II] examined in detail how they could be solved. Having at least one method for each type obviously means that we can successfully implement all the cases.

Table 2: The SPKI validity management methods

Type	Method	Speed of Revocation	Notes
A	No Validity Period / Only beginning time (= no end time)	N/A	Does NOT expire
B	End time / complete Validity Period	N/A	
	Renew	After current certificate expires	
C	CRL	After current CRL expires	
	Reval	After current "Bill of Health" expires	
D	One-time	Immediately	Can limit the usage of a group of users
E	Limit	Immediately	Can limit the usage of the particular user

The most interesting observations are related to choice between Types C and D, which are affected not only by the required speed of revocation, but also by the cost of implementing the system and providing sufficiently fast response time for the users.

Naturally, it would be nice to always use Type D as it makes the revocation immediate. Unfortunately, it also requires a network connection every time the certificate is used. This can be expensive to arrange compared to Type C method, which requires the network connection only at some intervals. Also, Type C is normally much faster to validate, because we do not have to access a server on the network. This can be an important factor in some situations, like implementing a ticket system for public transport, as each passenger cannot be forced to wait more than a moment for the validation or the access control system becomes a bottleneck for normal operation. Fortunately, as we concluded in [Publication II], the added risks of Type C method, which is the main reason for favouring Type D, can often be covered at least in

commercial systems by including the risk in the business model, just like credit card companies do.

Another interesting observation is related to choice between the different Type C methods. The CRL method works well when there is only one issuer, like in the public transport case. This finding is contrary to the finding in [Publication I] and reflects the author's current view. If on the other hand, there are many issuers, renew becomes an interesting option.

The other missing elements to solve the problem were the two missing protocols to actually use the methods: one to validate all the methods at the time the certificate is used, which was defined in [Publication I], and another to manage the validity status of a certificate (e.g. to revoke a certificate), which was defined in [Publication III].

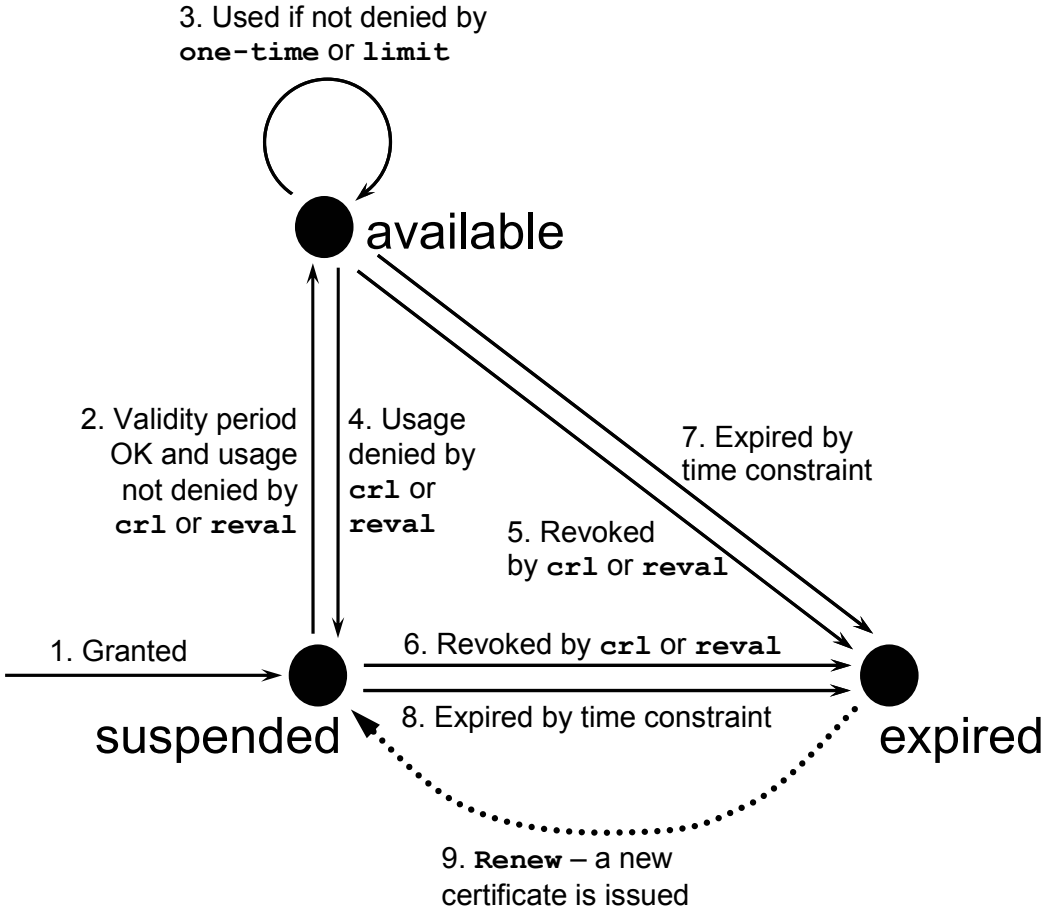

Figure 4: The Roles of Different Validation Methods

Now, with all the necessary elements it is possible to solve all the required use cases and hence, we have solved our research problem.

5.2. CHOOSING THE RIGHT VALIDITY METHOD

Based on the classification of methods, a heuristic for choosing the most suitable validation method for a particular situation has been developed and it is illustrated in Figure 5. Most, if not all, of these choices should be made by the designer of the system - they should not be left to the end users. We also note that using more than one online method in one certificate is usually redundant since the methods form a hierarchy, where the more capable can always achieve something a less capable method could. Therefore, the question is merely identifying what level is sufficient.

First we consider if the right depends on usage history, in which case the only option is Type E. Next, we consider if the right depends on context, in which case we take Type D. Type D is also the correct choice if the value of the right is such that we require revocation to take effect immediately and cannot compromise as discussed in the previous section. If however, there is room for compromise or the speed is not of the essence, Type B will suffice. However, it must be noted that for the end-users it is important that they do not need to bear the risk. The system should be such that from their point of view, the revocation takes place immediately.

Finally, unless we have a very good reason for choosing Type A, we should normally choose Type B. The reason is that Type A certificates do not expire and will therefore will remain in the system indefinitely – and if there are a lot of them, this will become a problem. Hence they should be used sparingly.

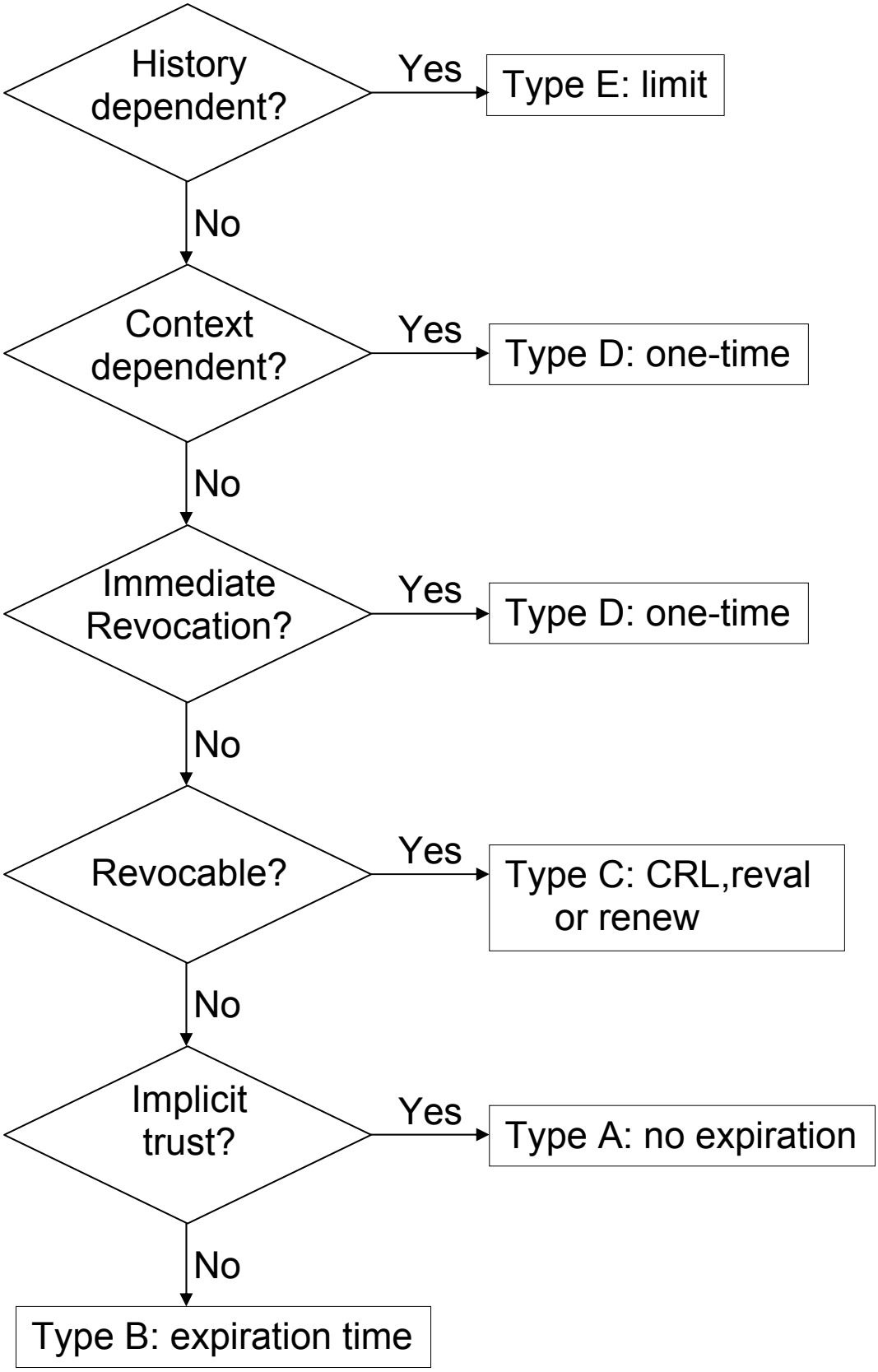

Figure 5: Heuristic for Choosing the Most Suitable Method

5.3. CRITERION: USABILITY

Managing access can be a challenging task for the system designers and resource owners, particularly if the system is large and distributed. Hence it becomes very important to make access control as easy to use as possible – only if users can correctly use the access control, can the system be secure and available as it is meant to be.

In a certificate-based system the users have essentially two different roles: the issuer and the subject. We have to assess the usability for both of them. As usability is very much a function of final system, not just the access control technology, we assess usability by evaluating the usability of our solution concepts to the cases.

Of the two roles, the subject is the less technical: they are using a system for some purpose (e.g. taking a bus to go shopping) and for them the access control is a necessary evil. To make it as usable as possible, it should not be represented as certificates or validation methods, but as something related to the application, in this case a bus ticket with an expiration time, or a phone number to call when the bus ticket or credit card is lost. Also, we note that it is relatively straight forward to transfer the remaining limit of credit to a new certificate if the earlier was lost.

Interestingly, the same applies to most issuers as well They too do not have to be aware of the technology, but can be presented with information that matches their goal: a button to cancel someone's right to use one's bank account. In all the examples we looked at in [Publication I] we discovered that a suitable goal-related form for the necessary information could be found and thus, extrapolate that the same should hold true for the majority of situations.

The designer of a system however, has to understand certificates and revocation methods. It is the designer's responsibility to choose a suitable method with which to support the user's goals and to analyse which options are relevant to the end-user.

A typical end user, e.g. someone using a certificate-based credit card, is less interested in the technical reasons for choosing between methods and more interested in the system behaving in an intuitive manner: when the parent presses the button to revoke the child's credit card, the revocation should take effect immediately, not after

some arbitrary time. For that reason, methods of Type C are not suitable: they sacrifice the sense of control for the benefit of reduced overhead. On the other hand, the delay does not have to matter to the end user – the possible misuse and its costs can be included in the business model of the system, similarly to the existing credit card systems, as we mentioned earlier.

Since one of the most interesting qualities of authorisation certificates is their support for delegation, we assume that many systems will make use of this feature and thus, force many users to make delegation decisions. Again, the solution is to use terms from the application domain: if we are delegating a credit card to a child, the user interface should talk about credit cards, not certificates.

In our evaluation we were unable to find any insurmountable obstacles for good usability: much of the usability rests on the system designer, but with careful design, good usability should be attainable. We therefore, conclude that this criteria was sufficiently fulfilled.

5.4. CRITERION: SECURITY

The security of SPKI authorisation certificates before the described changes has already been looked at in [14], so the focus of this evaluation is to look at the security implications the changes possess. We shall evaluate each of the three elements separately.

In [Publication I], the security of the validation protocol has been evaluated with respect to several key characteristics and we can conclude that it does not contain any significant problems in those respects. The security is naturally, dependent on the security of the transport mechanism used (the publication proposes using ISAKMP). The transport is responsible for authenticating parties and guaranteeing integrity and (optionally) privacy.

The management protocol [Publication III] is designed with the same principles, although it has not been evaluated in the publication. Again, its security is dependent on the security of the transport used. The protocol is very straightforward and hence,

is not likely to contain hidden problems. The one known issue is the lack of time stamps or sequence numbers in the commands, which can make it impossible to reconstruct the correct sequence in which commands were issued. This would become a problem if certificate issuer and validation server disagree on a reply sent by the validation server. Hence some method of ordering the commands should be added.

As to the new methods, renew is quite clear: conceptually it is very close to a Type B certificate with just the added online validation, which is as secure as other Type C online validations. Hence no new weaknesses.

Unfortunately, there still remains the issue related to the limit method. Limit was designed to prevent outsiders from targeting Denial of Service (DoS) attacks agains the subject by depleting the usage limit. It cannot however, prevent authorised chain members from making these attacks, as they are by design authorised to use the limit. They will however be caught, as any usage will require a signed request and is therefore traceable (all the online validations are designed so that an audit trail can be built during normal operation). Still, this means that a legitimate user could be unable to use some or all of her resource because of this type of misuse and the fact that the culprits can be traced might not make her happy at the time.

A more severe problem is that all online validations are vulnerable against a DoS attack aimed at the validation server: if the validity information cannot be fetched, the default interpretation is to regard the certificate invalid. Obviously, particularly D and E Types are affected by this, because these validations cannot be performed in advance. Unfortunately, as there is no general solution to DoS attacks, none exists for validation servers either.

This weakness could perhaps partially be softened by the verifier accepting at least small requests without the validation information and later performing the missed validations. This idea however, is left for Future work.

So, we can conclude that the first protocol has no known weaknesses, the second has only one, but the limit method has a few. Therefore, we can conclude that this criteria is mostly fulfilled.

5.5. CRITERION: SCALABILITY

The performance and scalability issues of certificate based systems in general and the validation protocol in particular still need further work. At the moment, they look promising, but without extensive empirical tests we can not state anything definite about their suitability as an Internet-wide solution.

Certificates are not very light to evaluate, as the evaluation entails verifying the signature, which is always a calculation intensive operation. Individual SPKI certificates are no more expensive to evaluate than any other certificates and in fact, if we compare evaluation of a single SPKI certificate to the combined evaluation of an identity certificate and an ACL entry, the latter is likely to be slower in all cases.

This problem will naturally become easier over time with increasing computational capacity and a modern desktop computer already verifies thousands of signatures in a second. A smaller device, like a PDA or a smart card on the other hand, is unable to perform signature verification fast enough to be usable in most applications without the help of a cryptographic coprocessor. These coprocessors can also be used in larger devices to boost performance without significantly adding to the complexity of the system. Another big factor is the choice of the signature algorithm, which have very different performance characteristics: the currently popular RSA[17] is heavier than upcoming options like Elliptic Curve[2] or NTRU[10].

The only party forced to evaluate large numbers of certificates is the verifier. In very large system, the verifier could be a cluster of computers, which raises the performance. (In clustering verifiers, the biggest question security-wise is that all the verifiers need access to the same private key.) Still, an upper limit for scalability exists, but should be high enough for most applications. After all, as global services are often created with several server clusters around the globe, each of these can have a dedicated access control cluster thus further improving scalability.

Another computationally heavy operation is the creation of new key pairs necessary for the subject wishing to remain anonymous. This operation is only necessary when receiving a new right, which is not likely to happen at a very high rate, so even a

moderately powered device can create the necessary keys by creating them in advance when the device is otherwise idle. So, even though the total number of keys created can be high, contrary to chain validation, this operation is spread to a very large device-base. Also, the choice of algorithm again plays an important role: Elliptic curves for instance, are significantly easier than RSA.

Finally, the creating of a certificate requires the creation of a signature, an operation that is computationally of the same magnitude than verifying. This operation is also spread among the user base, but not so effectively as key generation: some large service providers obviously create many more certificates for their users than individuals. Yet, these operations seldom have to be performed on small devices, so compared to validation, this is an easier problem.

We can conclude that the cryptographic operations required are not easy and they require some planning, but high scalability should be achievable. Further analysis is required however, for more exact conclusions.

SPKI draws many of its benefits from delegation and hence, distributed management of access rights. A result of this process is that the user could end up with a long chain of certificates that has to be presented whenever the right is used – and storing, handling and evaluating long chains can result in significant performance overhead. To solve this problem, the SPKI theory[7] introduces the concept of a *Chain Reduction Certificate* (*CRC*) – it is a certificate that corresponds to the semantics of the underlying certificates and online test results. Throughout this discussion it is important to bear in mind that at the moment, CRCs are merely an idea.

In creating CRCs, there are two options: all the online validations can be performed before reduction, in which case the resulting certificate has no online conditions, but presumably a shorter validity period. The other option is to include some or all of the online conditions in the CRC and let the verifier perform them as needed.

The main motivation for creating CRCs is performance benefits: by using a CRC, we can avoid repeating the costly operations of evaluating long chains and the validator can instead evaluate a single certificate to reach the access decision. But the use of

online limitations makes performance enhancements more difficult: it is possible to get more processing power to reduce the chain processing time by spending money, but no amount of money can reduce the inherent delays in communication networks.

Another motivation for creating CRCs is to promote anonymity by hiding parties in the chain. However, if a reduction certificate contains online checks, anonymity might be compromised. Therefore, any online validation does not appear to be compatible with reduction certificates created for privacy purposes. If on the other hand, the online validations can be performed before reduction and the resulting certificate has no online checks, the reduction might end up improving privacy. The problems of CRCs are looked at more closely in [Publication IV] and present highly interesting Future work.

We can conclude that the verifier is indeed a bottleneck in the system, but CRCs should provide partial relief.

Another potential bottleneck are the validation servers. Particularly in systems with large user bases, if a certificate high up in the certificate tree has an online validation (in particular: a limit validation), the server responsible for this validation is definitely a bottleneck. The fact that the validation servers can be selected freely and there can be several servers for each validity condition, helps significantly the scalability of the system. Still, this issue merits further work, because it poses a limit to the size of usable certificate trees.

The protocols presented in this thesis do not have any scalability issues that would significantly change the picture. Naturally, a secure transport requires the use of cryptography, but the additional load is of the same magnitude than chain evaluation, and the same remedies apply.

We can now conclude that of the three criteria, scalability remains the most unresolved. The certificate based system appears very scalable, but the online validations require some further work to be usable in very large systems. The developed protocols do not change the situation in a significant way. The limit method however, is the heaviest online validation, so any scalability problem in online validations is

bound to have a particularly strong effect with limit. Therefore, we conclude that this criteria is fulfilled only partially.

5.6. IMPLEMENTING THESE TECHNOLOGIES

The strengths of authorisation certificate based access control are in large distributed systems that require delegation, distributed management and anonymity. If these qualifiers do not apply, a centralised solution like ACL can be a more suitable solution. Still, this leaves us a large application domain, where authorisation certificates should be the correct choice.

Currently, authorisation certificates have practically no foothold. This is most likely the result of two main factors: the standards are not finished and as there are practically no existing solutions, no-one wants to be first to risk their operation on a new technology, particularly if they have to co-operate with other systems having traditional access control.

Still, after the standards are finished, introducing this technology is relatively easy. It does not require the existence of a large certification hierarchy, like identity certificates do, but if such exists, it can be utilised quite easily. The main problems are likely related to the devices used in the system: they have to have enough computational power and secure storage for the private keys. Still, these requirements are common to identity certificates, so undoubtedly solutions are being constructed.

5.7. A SUMMARY

I have successfully defined the missing elements in SPKI and designed elements so that the described use cases can be implemented and hence, the research problem solved. Of the three criteria, usability was sufficiently fulfilled, security was mostly fulfilled, but scalability remains only partially fulfilled.

6. FUTURE WORK

The main effort in future work should be directed in improving the scalability of the system. One promising way are the CRCs.

There are problems in creating CRC from chains that have online validations. It is not possible to perform all online validations in advance of usage. CRL and reval can be performed in advance - their result is a validity period which can be used to determine the validity period of the CRC. One-time and limit on the other hand, have to be evaluated at the time of usage and therefore they have to be included in the CRC. Finally, due to the nature of limit, it is not possible to perform a reduction over a certificate containing a limit condition, because that particular certificate has to be in the chain for the limit check to work. So, the problem remains for how online validations and limit in particular are handled.

Further performance improvements could be achieved, if all the remaining online validations in a CRC could be replaced with a single online validation representing all of them. Naturally, this raises trust issues, but could provide significant improvements, particularly in situation with limited network access. Another idea worth exploring is to use tokens: when the limit server is accessed, it grants the user tokens for more than one usage. Now, the user can utilise these tokens without further consultation with the limit-server. Naturally, this also introduces a delay in the revocation capability, but the amount of tokens could be balanced with the need for timely revocation providing us with usable compromise.

The treatment of multiple online validations in a chain seems to have natural links to the scalability problems discussed earlier. It might be particularly advantageous to

combine chain reduction with token fetching, thus creating certificates with no online validation but the requirement to present a suitable token.

7. CONCLUSIONS

In a distributed system, traditional centralised access control solutions present problems with scalability and anonymity. Proposed alternatives, like the SPKI authorisation certificate based access control is naturally a better fit to a distributed system. It facilitates the granting of rights by supporting delegation and distributed management. On the other hand, the problems of limiting usage or revoking the rights become more difficult, as the issuer of the right is no longer in control of the issued certificate.

In this thesis, we have discussed the problems of managing the online validation and revocation of SPKI authorisation certificates. All the existing solutions to these problems are based on online servers that give authoritative statements about the validity of a certificate. We have discussed the advantages and drawbacks of the various solutions and proposed two new ones. We have also presented a protocol for validating the certificates at the time they are used and, finally, we have presented a protocol for managing the online servers.

With these extensions, the SPKI certificates can now be used to implement many everyday applications, which were earlier impossible for SPKI. These include bus tickets worth 10 trips and credit cards with monthly limits. We have also concluded that the presented solution can be made usable and secure, but the scalability to global applications still requires further work.

We have discussed the role of chain reduction certificates as a possible element in achieving better scalability. We conclude that CRCs could provide performance improvements at minimal cost, if issued by the verifier. Finally, online validations still present challenges for reduction and should be examined further.

Possible future application areas for this technology include things like roaming in wireless networks and context aware applications.

REFERENCES

[1] Amorosi, E: Fundamentals of Computer Security Technology, Prentice Hall, New Jersey, 1994

[2] ANSI X9.62 - "Elliptic Curve Digital Signature Algorithm (ECDSA), 1999

[3] Atkins, D. et. al.: PGP Message Exchange Formats, Request for Comments: 1991, August 1996

[4] Berners-Lee, T., Fielding, R., Masinter, L.: Uniform Resource Identifiers (URI): Generic syntax, Request for Comments 2396, August 1998

[5] Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized Trust Management, In Proceedings of the 1996 EEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, May 1996

[6] Blaze, M., Feigenbaum, J., Ionnidis, J.: The KeyNote Trust-Management System Version 2, Request for Comments: 2704, September 1999

[7] Ellison, C.; Franz, B.; Lampson, B.; Rivest, R.; Thomas, B.; Ylönen, T.: SPKI certificate theory. Request for Comments: 2693, September 1999.

[8] Ellison, C.; Franz, B.; Lampson, B.; Rivest, R.; Thomas, B.; Ylönen, T.: Simple public key certificate. Internet draft (expired), IETF SPKI Working Group, July 1999.

[9] Heikkilä, J, Laukka, M: SPKI based Solution to Anonymous Payment and Transaction Authorization, Proceedings of the 4th Nordic Workshop on Secure IT Systems, 1999, Kista, Sweden

[10] Hoffstein, J., Pipher, J., Silverman, J.: NTRU: A Ring-Based Public Key Cryptosystem, Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, J.P. Buhler (ed.), Lecture Notes in Computer Science 1423, Springer-Verlag, Berlin, 1998, 267-288.

[11] Housley, R. et. al.: Internet X.509 Public Key Infrastructure Certificate and CRL Profile. Request for Comments: 2459, January 1999

[12] Naor, M., Nissim, K.: Certificate revocation and certificate update. In Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas, January 1998

[13] Naur, P. (ed.): Revised Report on the Algorithmic Language ALGOL 60, Communications of the ACM, Vol. 3 No.5, pp. 299-314, May 1960.

[14] Nikander, P.: An Architecture for Authorisation and Delegation in Distributed Object-Oriented Agent Systems, Doctoral dissertation, Helsinki University of Technology, 1999

[15] Rivest, R.: Can we eliminate certificate revocation lists? In Proceedings of the Second International Conference on Financial Cryptography, Anguilla, British West Indies, February 1998

[16] Rivest, R, Lampson, B.: SDSI – A simple distributed security infrastructure. In Proceedings on the 1996 USENIX Security Symposium, 1996

[17] Rivest, R., Shamir, A., Adleman, L.M.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM, v.21, n.2, Feb. 1978

PUBLICATIONS

A Revocation, Validation and Authentication Protocol for SPKI Based Delegation Systems *

Yki Kortesniemi
Helsinki University of Technology,
Department of Computer Science,
FIN-02015 HUT, Espoo, Finland
yki.kortesniemi@hut.fi

Tero Hasu
Helsinki University of Technology,
Department of Computer Science,
FIN-02015 HUT, Espoo, Finland
tero.hasu@hut.fi

Jonna Särs
Nixu Oy
Mäkelänkatu 91
FIN-00610 Helsinki, Finland
jonna.sars@nixu.fi

Abstract

In distributed systems, the access control mechanism is often modeled after stand-alone solutions, such as ACLs. Such arrangement, however, is not ideal as the system may be mirrored around the world and maintaining the ACLs becomes a problem. A new approach to this problem is using authorisation certificates to control access to resources. This diminishes management overhead, but introduces problems with revocation.

A related problem is enforcing quotas in distributed systems. Traditionally, authorisation certificates just limit the usage interval, but not the volume. In this paper, we discuss these problems in SPKI based delegation systems and propose some refinements to the SPKI specification. In particular, we address the problem of limiting the usage of resources to which a certificate grants access. Finally, we develop a protocol for solving these problems using online revocation and validation.

1 Introduction

Interactions between entities like people, organisations and software often rely on trust. If we trust someone or something, we are willing to grant them extra rights, and, similarly, we might receive some ourselves, if they trust us. However, to convince others that someone trusts us and has granted us the rights, we have to be able to provide them with some proof of this trust. To take an example, we might have a credit account in a financing company, which allows us to pay for our purchases and later settle our bill with the financing company. Here, the company trusts us to take care of the bill. However, to convince the merchant, we need proof this trust, which is expressed in the form of a card. The possession of a card assigned to the bearer is considered an assurance of this trust.

The use of a card was a good choice for on-the-place purchases as it made it relatively difficult for the majority of people to forge these expressions of trust. However, the situation is completely different when shopping over the Internet, or any other telecommunications line, where it is impossible to verify the possession of the card. Then, the right to purchase is granted solely based on the knowledge of the contents of the card, which makes it too easy to just copy this information and misuse it.

A more secure way would be to express the trust in the form of a certificate. A certificate is a digital document, where a signature is used to guarantee the unmodifiability of the information within. Certificates have traditionally appeared in two major types: identity certificates, where a trusted third-party testifies his belief that a particular public key belongs to a certain individual or other entity, and authorisation certificates, which grant some rights to the specified public key. The use of the right can be controlled by requiring the possession of the corresponding private key. A credit card can now relatively easily be represented as

*This work was partly funded by the TeSSA research project at Helsinki University of Technology under a grant from Tekes.

an authorisation certificate. Furthermore, the possession of this certificate alone will not grant any rights; the user of the certificate also has to prove the possession of the private key.

All of this can be accomplished with existing authorisation certificates, like the SPKI [6] certificates currently being standardised by the IETF. However, to solve anything other than the most trivial problems, the certificates alone will not suffice – we need other supporting mechanisms. First, it is necessary to have a mechanism to revoke a certificate in case of misuse, change of situation or if the private key is compromised. Second, we might want to impose some limits on the use of a certificate, like a monthly limit on purchases. This can not be accomplished with the certificate alone, additional online checking is necessary. Finally, we require a mechanism to authenticate the user of a certificate to make sure they possess the private key.

The revocation and validation problems have been discussed in the SPKI theory specification, but the structural specification leaves many of the details as well as the questions regarding a suitable protocol to accomplish these tasks completely unanswered. In this paper, we discuss the problems of revocation and validation and propose some refinements to the SPKI specification. Further, we present a protocol for online validation and revocation. As the protocol requires a communications channel that can guarantee integrity and authentication, we have based it on the ISAKMP [11] framework. ISAKMP is meant for providing a secure channel for key agreement, but can easily be extended to support other negotiations as well. It should be noted that some of the communication does not require ISAKMP and can use other, lighter protocols, as well, to reduce the overhead.

The rest of this paper is organised as follows: in Section 2 we explore these problems in detail and introduce our example case: the application of certificates as a replacement for credit cards. In Section 3 we go over different solutions to revocation and validation and establish the criteria for a good solution. In Section 4 we introduce the SPKI certificates, discuss their solution to revocation and validation, and finally suggest some refinements. In Section 5 we introduce the ISAKMP protocol, which we use as part of our solution. In Section 6 we detail the design rationale. In Section 7 we present our solution to the presented problems in the form of a protocol. Section 8 gives an example of the use of this protocol, Section 9 compares the solution to the criteria introduced and Section 10 discusses the limitations of these solutions and suggests further work. Finally, Section 11 presents our conclusions.

2 Problems in detail

In our example, we wanted to purchase something from the merchant over the Internet. In this case the various parties have the following interests during the transaction:

- The merchant is interested in
  - receiving the payment for his product or service
- We, on the other hand, want to make sure that
  - the payment is indeed received by the merchant and not by an imposter
  - the merchant is not able to charge us more than once
- We could also be interested in
  - hiding our identity from the merchant to avoid receiving further publicity material or because we do not want the merchant to keep a record of our purchases
- The credit company (or anyone else, who grants or delegates rights) might be interested in
  - controlling our purchases by imposing a monthly limit on our credit card
  - being able to cancel the card, should we misuse it or should the card fall in wrong hands

Now, how can these problems be solved using certificates and what new problems does this introduce?

The information contained in a credit card can easily be included in a certificate. However, not all the information is necessary if certificates are used. Credit cards normally have the name of the owner printed on the card to facilitate the verification that the card is indeed in the possession of its rightful owner. Certificates like the SPKI authorisation certificates, on the other hand, do not require the use of a name, because the use of a public/private key pair can accomplish the same end even better. An added bonus is that now the certificate only contains a public key, which does not identify the user, thus improving privacy. In fact, it is possible to use a different public key for every certificate, making it virtually impossible for the merchant to identify the user as long as the certificate is acquired from a source that does not reveal this information. The credit card company, however, has to know the identity of the holder of the card to be able to bill him.

In this respect, there is no change from the current situation and the credit company is still able to practice some data mining.

One additional advantage of a certificate compared to a credit card is delegation. It is possible to grant someone else a part or all of our own rights without giving away our own proof to these rights. An example could be that we want to allow our offspring to use our credit account. Now, with a traditional credit card, it would be possible (but usually not allowed) to loan our own card, thus losing control over it for a while. Further, if the offspring misuses our card, it is not possible to identify him as the guilty one. Another solution is to acquire a parallel card, but this requires a visit to the bank and has several limitations. If, however, the offspring uses his own certificate, which we delegated, it is possible to identify who has actually used the credit. And finally, the use of delegation does not require the credit company to be involved; we can use it at our convenience.

A more complicated, yet more realistic example could be that we want to allow our offspring to use our credit account, but only to a certain limit. A certificate alone can not accomplish this, as the certificate can not contain any information about its usage history. The certificate can not be modified to signify that it has been used and the information about the use can not be stored separately, as this additional information might "accidentally" get lost, should the user need more credit. One solution is to use an online server that keeps track of the amount of purchases. The certificate would then contain a reference to an obligatory online check that grants or denies every purchase based on the accumulated total. The use of an online server deviates from the basic idea that certificates are selfcontained and that they can be used without additional information. However, in many situations it is unrealistic to expect that certificates can be used without some additional control – certificates merely grant the right to use some resource, but used alone offer no solution (other than time periods) to control the volume of usage. Accepting the use of online validation opens up new possibilites in this area.

The choice of the validation server is up to the issuer of the certificate, and should be made so that the server understands the concept of limits. There could be different kinds of servers, some with more advanced capabilities like limit checking. Others, with less capabilities, can take care of simple problems, like verifying whether a certificate has been revoked or not. Revocation could result for instance from the compromise of the private key controlling the use of the certificate. Again, this feature requires that a revocation check is included in the certificate.

The online checking system is complicated because the entity verifying the certificate chain with online checks is usually the originator of the chain. The problem is that not all online checks can be allowed to be performed by everyone. With revocation, it is plausible that anyone can be allowed to verify whether a particular certificate has been revoked or not. However, when we talk about a limit type of check (a certain amount every month, a certain number of times, etc.), every successful validation also consumes part or all of the right to the limited resource. Therefore, only a party, to whom this limited use of resource has been granted either directly or through delegation, can be regarded to have the right to make these validation requests. Otherwise it would be possible for a malicious neighbour to use all of the limit (but not the resource itself) without the rightful parties' consent. Now, the verifier, being the originator of the chain, is neither the receiver of the limited resource nor his descendant. Therefore, for the validation to succeed, the user of the resource also has to delegate the right to make the validation check to the verifier using a validation certificate.

The remaining problem is, how to guarantee that the merchant receives one and only one payment. To accomplish this, the user of the credit card would delegate the merchant the right to charge his account by a specified amount and control the number of uses with an online check. This online check could be directed to the user's own terminal, which would eventually show that the merchant is requesting to use his right. The terminal could then validate this request once, and later deny any further validation attempts.

The merchant could, after having received the payment certificate from the user, contact the credit company, which can verify the certificate chain and, should the chain prove valid, credit the merchant's account. If the chain is not valid, the verifier can notify the merchant, which can then deny delivering the service. The merchant, on the other hand, can not deny having received the payment, and will therefore be caught, should he try to deny the service on the pretense of not being paid.

In all of these validation situations, it is paramount that all the parties in the negotiations are reliably authenticated to avoid any possible impersonations. First, it is important to verify that the validation server is indeed the intended server. This can be achieved by incorporating the server's public key in the validation part of the certificate and then using a suitable authentication mechanism. Further, the server has to verify that the party requesting the validation has been au-

thorised to perform it by verifying the certificate chain. Also, the merchant and credit card company have to authenticate each other to make sure the transaction happens to the benefit of the right parties.

The traditional SPKI view has been that the revocation information is fetched by the prover (in this case, the merchant or client). However, it may be impossible to equip clients having very limited computing and storage capabilities with the logic needed to acquire certificate chains. One solution would be that the verifier could also take care of completing the chains. The downside is that the verifier could face excessive loads, even denial of service attacks. Therefore, the verifier always has the right to refuse from anything other than verifying the chain and performing those checks the prover can not take care of. Another solution would be to introduce third party services for resolving the chains.

As a final point, it should be noted that arranging for reliable and efficient certificate revocation is difficult, no matter how good the protocols used are. Whenever possible, revocation should, therefore, be avoided altogether, by setting the validity periods of certificates small enough, so that if a potential problem with a certificate is noticed, any damages sustained by the time the certificate expires cannot climb too high. Revocation can be further obviated by choosing the policy of the verifier suitably. For example, the verifier could maintain a list of "problematic" entities, whose appearance in a certificate chain would cause the verifier not to accept the chain regardless of its validity.

3 Certificate revocation and validation

Certificates are designed to be self-contained, so that only a minimum amount of context information is needed to process the certificate. However, as mentioned before, the certificates cannot be completely independent. The trust relationships may change over time, while the information on the certificates still reflects the old circumstances. Thus, certificates may not live forever.

Revocation of certificates is always difficult, and especially so in systems like SPKI, where certificates are delegated among autonomous users for which there exists no centralised authority that could restrict delegation. Furthermore, in decentralised systems, traditional "operating system" style mechanisms such as simple deletion of the certificate [21] cannot be used to implement revocation, because there may exist multiple copies that we do not know of. [9]

Certificate revocation is intimately tied to the validity period and permission granted by the certificate. One key idea has been that certificates are only valid for a reasonably short period or grant a limited permission. Then, the loss would be limited, should the private key be compromised, and other precautions, like revocation, would be less critical. [19] Maybe they could even be omitted. This would be desirable, as a revocation check every time a certificate is used can amount to significant traffic. If, however, it is impractical to use very "short" certificates, revocation can become necessary.

Different revocation mechanisms can be evaluated according to certain properties: *timeliness*, *third party side effects*, *reversal of revocation*, and *granularity*. Of these properties, granularity and timeliness are the most important. [1] In addition, some revocation mechanisms *protect* the ability to revoke certificates so that only the issuer or the certificate owner has a right to revoke it. [4]

Further design criteria for revocation could be that the revocation mechanism should provide *fail safety* and *availability*. Also, it should be *recent, adjustable and bounded* in terms of revocation delays and *contained* so that compromises in the revocation do not allow further compromises of the system. [20]

3.1 Validity periods

The basic method for limiting certificate validity, which most certificate types have in common, are validity period dates. They are often called the "not before" date and the "not after" date.

Validity periods are easy and efficient to check, even in an offline environment, but they also have drawbacks. The need to revoke a certificate may arise long before the certificate was originally planned to become outdated. The longer life span the certificate has, the longer is the potential period during which the certificate is spreading false information. Thus, if a validity period is used as the only validation mechanism in a certificate, the period should be specified as short as possible. [19]

If certificates with very short validity periods are used, the management overhead might easily grow too large. To reduce the overhead, the certificates could indicate a location from where a replacement certificate can be fetched. If the information in the certificate is still valid, the replacement can be issued as a standard procedure.

3.2 Certificate Revocation Lists

CRLs are the most common revocation method used in combination with validity periods. A CRL is a signed

list issued by the Certificate Issuer identifying all revoked certificates by their serial numbers or some other reliable identification. If the certificate is not on the list, it is assumed valid. The list includes a time stamp or a validity period. The CRLs are published on a periodic basis, even if there are no changes, to prevent replaying old CRLs. [14]

The main problem with CRLs is that they only shorten the period of possibly false information taken as correct, but they do not eliminate it. Further more, the verifier has no control over how often the CRL is updated, and thus cannot affect the amount of risk it is accepting [16]. The CRLs also may get very long, requiring a lot of bandwidth, a large storage capacity and excessive processing.

There have been several proposals for improving the performance of the CRLs [14]. Some of the most accepted are using short validity periods for certificates in the first place, thus shortening the time the certificates spend on the CRL, and using Delta-CRLs that only include the changes since last update instead of sending the complete list every time. To complicate matters, some techniques to improve the performance have been patented. [12]

Essentially, CRLs are a memory from the age of manually verifying credit cards. Today, when even refrigerators are going online, it could be argued that a more online-oriented solution could be used.

3.3 Certificate Revocation Trees

One proposed solution to the revocation problem is called a Certificate Revocation Tree (CRT) [14]. A CRT issuer creates a group of statements of the type "If the CA is X and the serial number is between Y and Z, the certificate is valid". Together, the group specifies the status of any certificate known by the issuer. These statements are placed as leafs in a binary tree structure and the tree nodes are filled with hash values calculated from the child nodes. Finally, the root node value is signed by the issuer to provide proof of integrity.

To check the validity of a certificate, the verifier needs to check the appropriate statement, and verify the associated hash values and the root node's signature. The other statements and hash values do not need to be transfered nor stored. However, the tree must be completely rebuilt and signed every time the status of any single certificate changes.

3.4 Online validation

If all the parties can be assumed to stay online, the most simple, efficient and timely way for the verifier to check revocation is to directly ask the issuer or a validity server about the certificate in question. The issuer or validity server may respond with a simple boolean value together with a timestamp and a signature, or the reply may also include other information such as a time period when no further proof of validity is required.

An alternative solution based on regularly sent affirmation tokens has been proposed [7]. If this token is not received in time, the certificate is taken as having been revoked. However, this approach fails to consider communications disruptions. Also, it requires a global clock, which is not a practical notion in a world wide distributed environment. Rivest comes to a similar idea of using positive affirmations in his analysis of CRL. [16]

Although the online check seems very simple, it is flexible enough to allow for a wide variety of validation policies. The validation server could simply say the certificate is valid if it has not been revoked, but it could also keep track of the context of how many times and how the user has used the certificate, and make the validation decisions based on the context.

Online validation is simple for the verifier, but requires more processing power from the validation server, who must create a signature for each reply.

The general opinion seems to be moving from CRLs to online checks. The X.509 specification has originally relied on CRLs. However, there is a draft that defines an online status protocol similar to the one we are proposing. [13]

4 SPKI certificates, validation and revocation

Simple Public Key Infrastructure (SPKI) is a proposal for a Public Key Infrastructure (PKI) that would be more flexible than X.509 and free from the requirement of a global, trusted Certification Authority hierarchy. It has adopted many ideas from the SDSI [18, 17] and PolicyMaker [3] prototype systems. IETF is developing SPKI, and so far it has reached the experimental status.

SPKI was designed to support certificate based authorisation. It can be used to certify identity, as well, but unlike X.509 and other name oriented systems, SPKI uses cryptographic keys to represent identities. To facilitate certificate management by humans, SPKI has local name spaces that can be linked together.

SPKI authorisation certificates [5], like any authorisation certificates, are signed statements of authorisation. The certificate can be abstracted into a signed quintuple (I, S, D, A, V) where

I is the Issuer's (signer's) public key, or a secure hash of the public key,

S is the Subject of the certificate, typically a public key, a secure hash of a public key, a name, or a secure hash of some object,

D is a Delegation bit,

A is the Authorisation field, describing what access rights the Issuer delegates to the Subject,

V is a Validation field, decscribing the conditions (such as a time range) under which the certificate can be considered valid.

The meaning of an SPKI authorisation certificate can be stated as follows:

Based on the assumption that I has the control over the rights or other information described in A, I grants S the rights/property A whenever V is true. Furthermore, if D is true and S is not a hash of an object, S may further delegate A or any subset of it.

4.1 SPKI validity conditions

SPKI certificates, like most other certificate types, have a validity period. In SPKI, the validity period definition consists of two parts:

```
<not-before>:: "(" "not-before" <date> ")"
               ;
<not-after>:: "(" "not-after" <date> ")" ;
```

Both parts are optional and if either one is missing, the certificate is assumed to be valid for all time in that direction. There is an additional type of validity period called "now", which has a length of 0. It can only be the result of an online check and is interpreted to mean that the certificate is valid the moment the validation request was made, but it states nothing about the future. If the same certificate is used repeatedly, the online check has to be repeated, as well.

In addition to the validity period, SPKI includes three online validity checks: CRLs, revalidations and one-time checks. Furthermore, the SPKI theory [6] defines other online checks, but they do not appear in the structure drafts [5], yet. Later in this paper we discuss and propose structures and reply formats for some of them.

To facilitate the desision of whether or not the certificate is valid at a particular instance of time, all the different validity conditions end up being converted to validity periods as specified above. So, validating a certificate is relatively straightforward: check that the validity period stated in the certificate as well as the online checks (converted to validity periods) are all valid at the time of use and the certificate as a whole is valid and, therefore, grants the included permission.

4.2 SPKI online checks

All the online checks are defind using the following format:

```
<online-test>:: "(" "online" <online-type>
                <uris> <principal> <s-
                part>* ")" ;
```

where `<online-type>` can be `crl`, `reval` or `one-time`. The `<uris>` specify one or more URIs (Uniform Resource Identifier [2]) that can be used to request revalidation. The `<s-part>` is optional and may contain parameters to be used in the online check.

SPKI includes both traditional and delta CRLs in its specification. These must also be signed by the aforementioned principal. The CRL formats are specified below.

```
<crl>:: "(" "crl" <version>?  "(" "can-
        celed" <hash>* ")" <not-before>?
        <not-after>?  ")" ;

<delta-crl>:: "(" "delta-crl" <version>?
              <hash-of-crl> "(" "canceled"
              <hash>* ")" <not-before>?
              <not-after>?  ")" ;
```

Another way of getting assurance that the certificate is still valid is to ask for a "bill of health" which testifies that the certificate can be considered valid for the stated period. The SPKI definitions specify the reply format:

```
<reval>:: "(" "reval" <version>?  "("
          "cert" <hash> ")" <not-before>?
          <not-after>?  ")" ;
```

The reply identifies the original certificate in the hash and gives a confirmed validity period for that certificate. The reply must be signed with the key given as `<principal>` in the original certificate.

The third option is that the verifier of a certificate can just ask the issuer directly about the certificate's validity every time the certificate is used. In SPKI, this is called one-time validation, as the validation proof is valid one time only, at the moment the reply is received. The corresponding reply message is:

```
<reval>:: "(" "reval" <version>?  "("
          "cert" <hash> ")" "(" "one-time"
          <nonce> ")" ")";
```

Again, the hash must correspond to the original certificate, and the reply message must be signed by the principal given in the certificate.

4.3 Proposed changes to SPKI

In light of our earlier comments, we propose a number of changes to the SPKI structure.

Proposition 4.1 *Deprecate* `crl`.

In the SPKI context, CRLs are an outdated, impractical technology. They are at their best in situations where there are only few certificate issuers and it is thus possible to prefetch most or all relevant CRLs and then work offline. But in the SPKI model there are possibly a huge number of certificate issuers and it is not possible to predict, which CRLs are going to be used, so the online connection is still required. Furthermore, to validate a single certificate using CRLs, it is necessary to download a potentially long list of information, most of which is useless unless other certificates from the same issuer are validated in the near future.

A better way to manage revocation is to use `reval`, which provides only the necessary information about the certificate in question and nothing more. However, even better is to use short lived certificates and avoid online checks altogether.

Proposition 4.2 *Introduce online test query formats.*

```
<crl-query>:: "(" "test" <version>?  "crl"
              "forbid-delta"?  ")" ;
<reval-query>:: "(" "test" <version>?
                "reval" <cert> ")" ;
<one-time-query>:: "(" "test" <version>?
                   "one-time" <cert>
                   <nonce> ")" ;
<valid-basic>:: <valid-date> | <valid-
                dates> ;
<valid-date>:: <not-before> | <not-after> ;
<valid-dates>:: <not-before> <not-after> ;
```

Proposition 4.3 *Introduce negative online test reply formats for* `reval` *and* `one-time`.

The SPKI specification currently defines online test reply formats for tests of type `crl`, `reval` and `one-time`. However, the definitions for `reval` and `one-time` assume positive replies. To make it possible for a verifier to prove that a test failed, negative reply formats should also be defined. We propose the following reply formats, which support both positive and negative replies to `reval` and `one-time` queries, respectively.

```
<reval-reply>:: "(" "reval" <version>?  "("
                "cert" <hash> ")" "in-
                valid"?  <valid-basic> ")"
                ;
<one-time-reply>:: "(" "reval" <version>?
                   "(" "cert" <hash> ")"
                   "invalid"?  "(" "one-
                   time" <nonce> ")" ")"
                   ;
```

To allow use as proof, all replies must be digitally signed by the validator.

Proposition 4.4 *Introduce* `renew`.

The SPKI theory document states that SPKI has a mechanism to fetch a sequel to the current (short lived) certificate; this provides an alternative way of controlling revocation. As the specification itself does not currently define the format for this kind of online check or the related messages, we will propose such formats here.

```
<renew-test>:: "(" "online" "renew" <uris>
               <principal> <s-part>* ")" ;
<renew-query>:: "(" "test" <version>?  "re-
                new" <cert> ")" ;
<renew-reply>:: "(" "renew" <version>?
                <cert> ")" ;
<renew-reply>:: "(" "renew" <version>?  "("
                "cert" <hash> ")" <valid-
                basic>?  ")" ;
```

The former `<renew-reply>` is a positive reply, and contains the new certificate. The latter one is a negative reply, and contains the hash of the certificate for which an extension certificate was asked for. The validity period states a period of time during which renewal requests will be denied.

Proposition 4.5 *Introduce* `limit`.

Online tests guarding limited resources should be distinguished from other online tests and we propose a new type of an online check called `limit`. It is similar to `one-time`, but a verifier may not perform a `limit` check without proof of its right to ask about the validity of the certificate containing the test. Our proposals for the syntax of the test and the related messages are below.

```
<limit-test>:: "(" "online" "limit" <uris>
               <principal> <s-part>* ")" ;
<limit-query>:: "(" "test" <version>?
                "limit" <cert> <request>?
                <chain> ")" ;
```

where `<cert>` is the certificate whose online test(s) are to be made, `<request>` specifies the amount of resources requested, and `<chain>` proves that the verifier is entitled to ask about the validity of the certificate. The last certificate of the chain must be the validation certificate, which contains the `<nonce>` that is to be included in the reply to the query.

```
<request>:: "(" "request" <s-part> ")" ;
<chain>:: "(" "chain" <cert>+ ")" ;
<limit-reply>:: "(" "limit" <version>?  "("
                "cert" <hash> ")" "in-
                valid"?  "(" "one-time"
                <nonce> ")" <context> ")" ;
<context>:: "(" "context" <hash> ")" ;
```

where `<hash>` is a hash of the concatenation of the canonical forms of `<request>` and `<chain>`.

5 ISAKMP

The Internet Security Association and Key Management Protocol (ISAKMP) [11] has been designed to be a framework for securely implementing key and security association agreement negotiations. A security association (SA) is a simplex communication channel, which provides integrity, authentication and possibly confidentiality. The actual channel can be implemented using various techniques, like IPSec, but the role of the management protocol is to agree on the parameters used for the channel, such as the algorithms used. To provide high bandwith two-way communications, at least two different SAs (one in each direction) have to be agreed on.

ISAKMP provides the building blocks for defining the actual negotiation protocols by defining the types of information that can be passed between the negotiating parties and by defining a two-phase process for the negotiation. In this model, the first phase is used to agree on an internal SA, which is then used to protect the possibly numerous phase two negotiations. This makes the phase two negotiations much more simple as they do not have to worry about securing their communication. The phase two negotiations then agree on the parameters for the actual communications. These can include negotiations on an SA for the communication as well as the keys used.

A negotiation (be it a phase one or phase two negotiation) is described in the ISAKMP world as an exchange. The exchange defines the order and contents of the messages sent between parties. The ISAKMP RFC defines some exchanges, but the actual protocols are free to define their own.

One example of a key agreement protocol built on top of ISAKMP is the Internet Key Exchange (IKE) [8], which uses techniques from the Oakley [15] and SKEME [10] RFCs to define a key agreement protocol for the Internet environment. It uses two of the ISAKMP exchanges for its phase one negotiation and defines its own phase two negotiation.

In our case, we use the ISAKMP to define the negotiation protocols for validating the certificates and for using the rights granted by the certificate. ISAKMP is used to provide integrity and authentication and possibly confidentiality by using the standard ISAKMP phase one exchanges to create a suitable SA. We then define new phase two exchanges for the negotiations.

Even though our protocols are not key agreement nor SA negotiations, the use of ISAKMP can be justified because they share many similar characteristics. Further, the use of ISAKMP makes the protocol more secure as ISAKMP takes care of most of the security problems. And finally, this makes the implementation of the protocol easier, as most of the protocol functionality is already implemented in ISAKMP.

The actual communications in our protocol may involve three or more parties, so a three-party protocol could possibly be even more suitable than ISAKMP. However, the evaluation of this option is left to future work.

6 Background for the protocol

In this section we go over some of the essential problems in implementing a validation protocol.

6.1 The SPKI reality

For some applications revocation is essential. In SPKI, revocation and online validation is possible only if it has been defined in the certificate. The format of an online check definition was already described in Section 4.2 for those online tests currently included in the SPKI specification. The goal for our protocol was to support them, as well as the tests proposed in Section 4.3.

An online check expression must contain one or more URIs. The purpose of an URI is to define the protocol used to perform the verification, and to identify the entity or resource that should be consulted using the protocol. Only one of the URIs should be chosen and used during validation, and the others should be considered as backups in case the initially chosen entity or resource cannot be reached. The `<principal>` field is used to authenticate the server; it typically contains the public key of the server. The `<s-part>*` part of an online check definition may contain additional information that only needs to be understood by the validation server. Depending on the type of URI, the same information could also be contained in the URI itself. (This is true for an HTTP URL, at least.)

Once a verifier receives a certificate chain, it must first check to see if the chain is valid, apart from the online checks. It may be that only the verifier is able

to understand the tags in the certificates. Only if the chain is otherwise valid should the verifier proceed to make the online checks.

6.2 Authenticating the parties involved

A successful validation depends on several things. First, we have to be able to authenticate the participants or the source of information reliably. The SPKI specification does not give details regarding connecting to online servers or transmitting messages between them. One way to solve the problem is to use ISAKMP to authenticate the parties. The relevant public keys can be found in the certificate chain: the verifier is the originator of the chain and the possible validator is identified in the validation part of the certificate requiring online validation. As both parties know each others' public keys and have their own private keys, authentication and possible session key exchange can be arranged. Our protocol requires authenticity and integrity from the security association; other qualities, such as confidentiality are optional, and are left for application specific policies to decide.

It should be noted that it is not necessary to authenticate the parties in every transaction type. For instance, while fetching a CRL, it is not necessary to authenticate the parties involved as long as the CRL is correctly signed. In such cases, ISAKMP can be limited to first part of the protocol, namely the service request.

6.3 Authorising the limit validations

The second problem is related to the right to make some validation requests: validity queries of type `crl`, `reval`, `one-time` and `renew` do not diminish any limited resource and can therefore be made by anyone. A `limit`-type validation, however, will use some or all of the resource by approving the validation, and therefore the access to such validation has to be limited to only those who are able to use the related resource themselves. In practice, this means those entities, to whom the limited right was granted, and all other entities, to whom this right was further delegated.

As the verifier is not a receiver of the right, but rather the originator of the chain, he must not be allowed to make any `limit`-type checks in the chain without explicit permission. The user of the resource, i.e. the final receiver in the chain, has to authorise the verifier to validate the certificates by issuing a special validation certificate for this particular use of this particular chain. In our example, the merchant would authorise the credit card company to make all the necessary online checks.

6.4 Auditing the validations

All of the online checks in a certificate chain must pass; otherwise the certificate chain is not valid. It is in the chain verifier's best interest that it handles the verification correctly, as it is usually guarding access to its own resource. In any case, the verifier should be the one responsible for properly verifying the chain. It could be argued that the verifier must also be able to prove that it verified a chain according to the rules in case some in the chain denies having authorised the transaction by having revoked one or more certificates. However, the need for proofs depend on the nature of the service and is therefore a policy decision.

It is possible for the verifier to have proof if it stores the verified chain, as well as the signed replies sent by the validation servers mentioned in the online checks. Now, the verifier should only approve a chain when it has such a signed statement for each of the online checks in the chain.

6.5 Validation certificates

A validation certificate must contain at least all the fields shown in certificate $c_{validation}$.

$$c_{validation} = \texttt{(cert (issuer } K_{issuer}\texttt{) (subject } K_{subject}\texttt{) (tag (validate } hash(S_{chain})\texttt{)) (nonce } v_{nonce}\texttt{)) (not-after } T_{expiration}\texttt{))} \quad (1)$$

where K_{issuer} is the public key of an entity authorised to issue a permission to validate certificate chain S_{chain}, $K_{subject}$ is the public key of an entity that wishes to check the validity of S_{chain} (i.e. the verifier). v_{nonce} should be a unique value in the sense that after the validation server has seen a certificate that has a particular v_{nonce} value, it will not accept another certificate with the same value until after the expiration time $T_{expiration}$ of the first certificate has been reached. $T_{expiration}$ should be chosen to provide sufficient time for validation, but nothing more.

6.6 Avoiding unnecessary checks

A possibility for a certificate to get unnecessarily used is when there are multiple `limit`-type online checks in a chain. If these limit checks are performed sequentially, it could be that some checks pass, before one of the checks fails thus wasting the limits checked so far. Now, all unlimited checks can then be performed first, and only after that should any of the limited-use checks be made.

In our case, we have used the refined SPKI specification, which gives us new options. In order to reduce the likelihood of wasted checks, we have decided

to use two-phase negotiation for `limit`-type validations and one-phase negotiation for others. Furthermore, the one-phase negotiations can be performed without an ISAKMP connection as the integrity of the information is not at risk. The two-phase negotiations, however, either require ISAKMP or signed requests and replies.

One possibility of unnecessary use of limited-use checks still remains. Any network failures during the second phase might cause the transaction to fail when it is already partially complete. As online checks cannot be cancelled, there is no possibility of rolling back the transaction, and those online checks already committed may have been wasted. To alleviate this situation, the implementation can try to recover by rereserving the resource and committing again. Also, the validation server can keep the reservations past the timeout until someone else reserves the resource. Then, if the network failure is temporary and verifier keeps sending the commit request even after the timeout (but still within the authorisation), the commit may succeed.

7 The SPKI Validation Protocol

An overview of the protocol from the verifier's point of view has been given in Figure 1. In the first phase, the validation servers are queried to see if the online checks would pass or not (see Figure 2). For non-limit validations, the final response will come already in this phase. For limit validations, if the replies to the queries indicate that all of the checks will pass, the verifier can then commence with the second phase, in which all the reservations are then committed (see Figure 3).

7.1 Message formats

Between client and verifier

All the messages in this section have been defined using expressions resembling S-expressions for uniformity and readability purposes. The actual messages will follow the ISAKMP message structure and an example of a message in ISAKMP form has been included. The conversion of other messages is equally straightforward.

When a client wants to request a service from a service provider, it sends a message containing the following information to the server:

(Message definition 1)

```
(service-request
  (version VERSION)          [optional]
  (request REQUEST)
  (auth CHAIN)
  (valid-auth VALIDCERT)     [optional]
  (verbose))                 [optional]
```

where `VERSION` is a byte string that uniquely defines the version of the message format. `REQUEST` is a free-form field understood by both the client and the server/verifier, `CHAIN` is the certificate chain proving that the client has the permission to request the service, and `VALIDCERT` is the validation certificate that proves that the verifier has the right to check all the limited-use online checks contained in `CHAIN`. `"verbose"` is an optional field that, if present, specifies that the verifier should give detailed error messages; instead of a single return value, the verifier should reply by sending the entire chain of certificates it attempted to verify and a reason code for each online check contained within those certificates. The possible reasoncodes are listed in Section 7.2.

The information contained in messages of the above format can be represented using ISAKMP payloads as illustrated in Figure 4.

Between verifier and validator, unlimited checks

All online checks except `limit` checks can be performed in one phase. For `crl`, `reval` and `one-time` checks, the verifier sends to the validator a request of the form:

(Message definition 2)

```
(validation-request
  (version VERSION)          [optional]
  (spki-query QUERY)
  (verbose))                 [optional]
```

where `QUERY` is a validation query as defined in Section 4.3.

The validator then sends back a reply of the form:

(Message definition 3)

```
(validation-reply
  (version VERSION)          [optional]
  (spki-query hash(QUERY))
  (spki-reply REPLY)
  (reason REASONCODES))
```

where `hash(QUERY)` is a reference to the query. `REPLY` is the reply as defined in Section 4.3.

Between verifier and validator, limited checks

`limit` checks have to be performed in two phases to make sure all the checks in the chain will either succeed or fail. In the first message the verifier announces the wish to use some of the limit:

(Message definition 4)

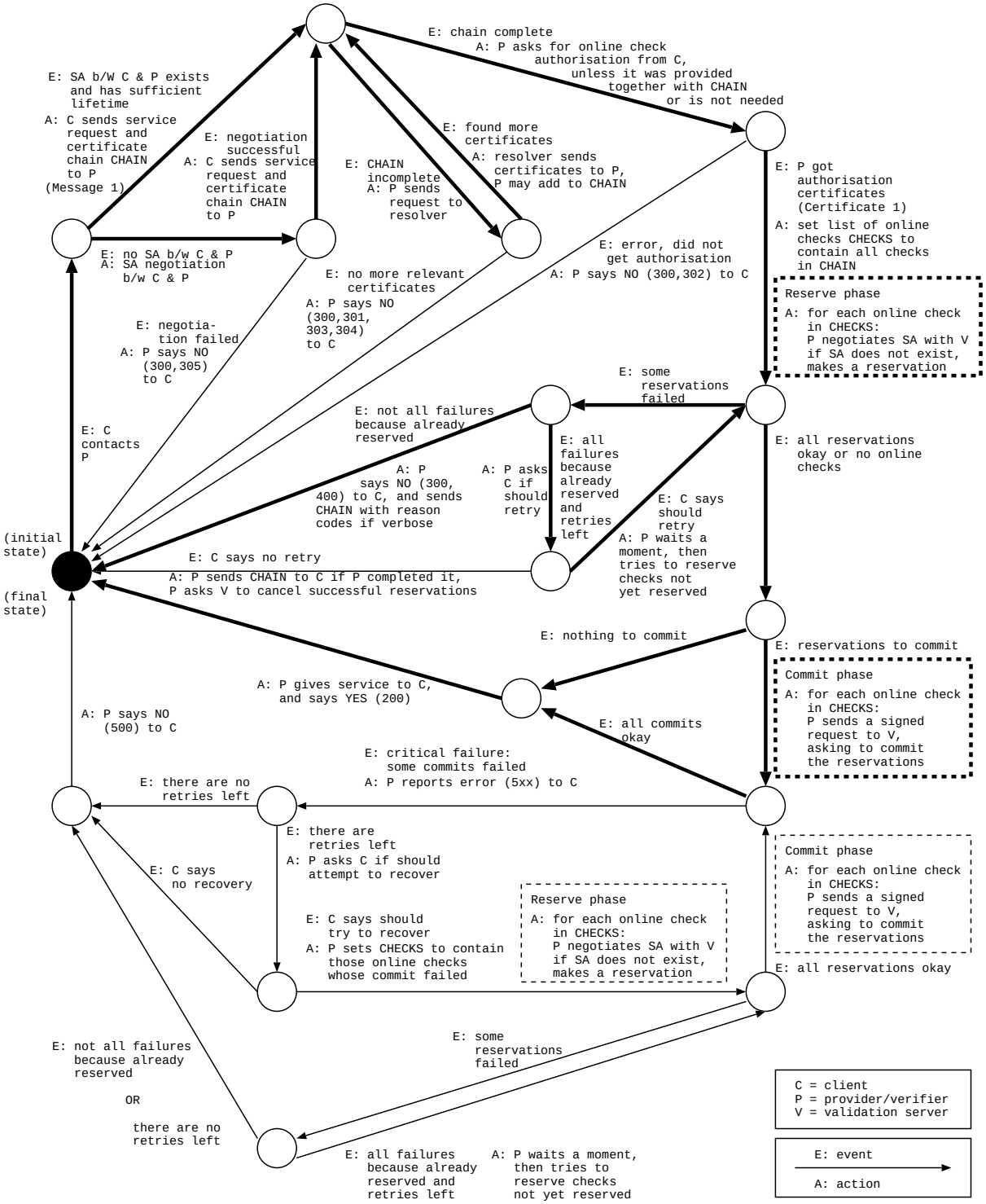

Figure 1: Verifier state machine.

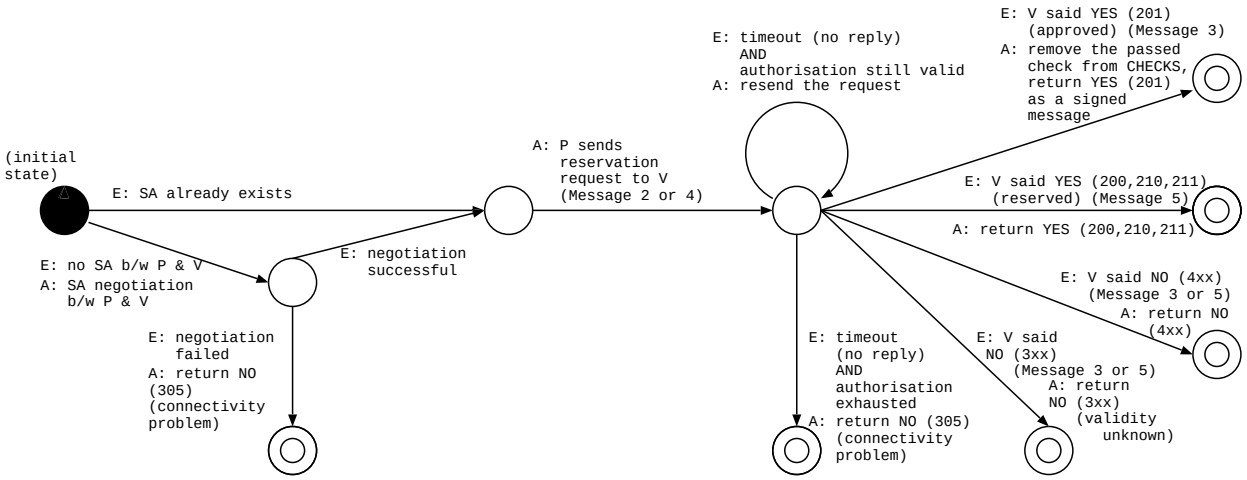

Figure 2: The reserve phase. Executed concurrently for each online check in the list of checks $CHECKS$.

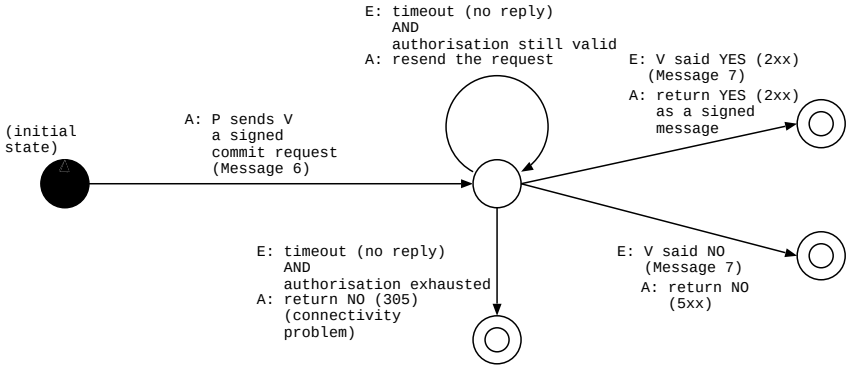

Figure 3: The commit phase. Executed concurrently for each online check in the list of checks $CHECKS$.

```
(reservation-request
  (version VERSION)           [optional]
  (spki-query QUERY)
  (verbose))                  [optional]
```

In the reply, the validator informs the verifier, whether the necessary limit exists:

(Message definition 5)

```
(reservation-reply
  (version VERSION)           [optional]
  (spki-query hash(QUERY))
  (reason REASONCODES)
  (commit-by COMMITBY))       [optional]
```

where `COMMITBY` indicates by which time the verifier has to confirm that it wants to use the limit. The validator has reserved the limit for the indicated time and if the verifier does not send the confirmation within the indicated timeframe, the reservation will expire.

This does present a problem for the protocol: if for some reason the verifier is unable to send the confirmation message in time although other confirmations in the chain have been sent, there is a risk that the chain will not be completely valid and some limits will be lost. The verifier can try to compensate by rereserving the limit, but this is only a partial solution. Further study of this problem is left to future work.

When the verifier has successfully reserved all the necessary limits, it can send the confirmation message:

(Message definition 6)

```
(commit-request
  (version VERSION)           [optional]
  (spki-query hash(QUERY))
  (cancel)                    [optional]
  (verbose))                  [optional]
```

where `"cancel"` indicates that the verifier does not want to confirm the reservation. This would be applicable if some other reservations had failed, for instance.

The validator will reply with a message of the form:

(Message definition 7)

```
(commit-reply
  (version VERSION)           [optional]
  (spki-query hash(QUERY))
  (spki-reply REPLY)
  (reason REASONCODES))
```

It should be noted that if the confirmation for some reason arrived late, the reply could be negative.

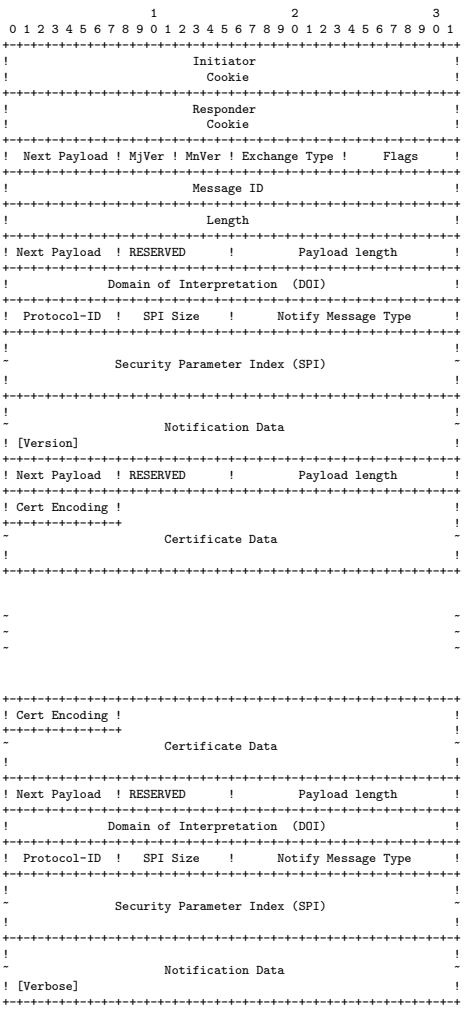

Figure 4: An ISAKMP payload definition of a service request.

7.2 Reason codes

The possible `REASONCODE` values are divided into categories as follows:

1xx Informational. These values are reserved for informational messages.

2xx Is valid.

- **200** YES, for no particular reason. The server did not specify a reason for saying yes.
- **201** YES, is valid. Indicates the resource was not reserved, and that the online check was already performed and it passed. This code should only get returned if the resource is of a non-exhaustible nature.

210 YES, reservation was successful. The online check was reserved and will be available for a commit for a limited period of time.

211 YES, reservation was committed, and the online check thus passed.

3xx Not known if valid.

300 NO, validity unknown for no particular reason. The server did not specify a reason for its inability/unwillingness to determine if the certificate is valid.

301 NO, try later. E.g., the resolver was busy and a complete chain could not be formed, or reservation would have been possible unless some other reservation(s) had not already been made.

302 NO, not authorised to ask. The authorisation provided was insufficient.

303 NO, send complete chain. The client should send the complete certificate chain to use. The server is not willing to acquire the chain for the client.

304 NO, incomplete certificate chain. The server tried to complete the chain provided by the client, but failed.

305 NO, connectivity problem.

310 NO, not interested. The server is not authorised to validate the certificate.

311 NO, syntax error. E.g., the validation server did not understand the question due to a malformed request.

4xx Not valid.

400 NO, for no particular reason. The server did not specify a reason for saying no.

401 NO, was revoked. The certificate has been revoked.

402 NO, is exhausted. The resource that the online check was guarding has been (possibly temporarily) exhausted.

5xx Severe error occurred. Some, but not all of the online checks were committed.

500 NO, a severe error has occurred. The server did not specify more details about the error.

501 NO, connectivity problem at a critical moment.

8 An example

As an example of the usage of our protocol we cover a scenario in which certificates are used to authorise and control credit-card-like payments, like in our original example.

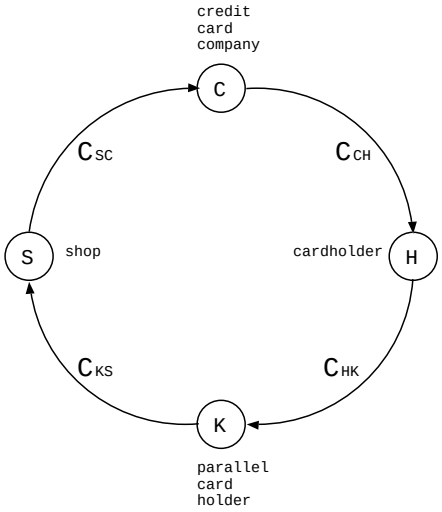

Figure 5: An example scenario.

In the scenario we have a credit card company C and a "cardholder" H. C issues H a certificate c_{CH} with which it authorises H to make payments, which will first be debited from C's account, and which H should later pay back to C. (This certificate represents a traditional credit card.)

c_{CH} = `(cert (issuer` C`) (subject` H`) (propagate) (tag (has-credit unlimited)) (not-after` E`))` **(2)**

where E is the expiration date. Here we are assuming that H has no monthly credit limit. It should be noted that the account number of H is not mentioned in the certificate. This is not necessary as the credit company can store this information, when it issues this certificate. The account number is of no concern to the user nor the merchant; in fact, leaving it out promotes privacy. Also, it makes it possible to change the account number and Certificate 2 without affecting any of the subsequent certificates in the chain.

H wants to give his offspring K a "parallel card", i.e. H wants to allow K to use his credit account. However, H does not trust K to fully understand the value of money, and wants to only allow K to accumulate a maximum of \$500 worth of debt to H. To do this H sets up a validation server (or uses an existing one), and issues the certificate c_{HK} to K.

c_{HK} = (cert (issuer H) (subject K) (propagate) (tag (has-credit \$500)) (not-after E) (online limit (uri svp:hv.net) H_v (max \$500))) **(3)**

where the URI prefix `svp:` (SPKI Validation Protocol) refers to the validation protocol presented in this paper. H_v is the principal that handles validation for H, and `hv.net` is the DNS domain name of H_v.

The validation server H_v keeps track of the transactions initiated by K, and will only confirm the validity of a certificate if that certificate does not cause the limit mentioned in the validity check field of c_{HK} to be exceeded.

The value \$500 in the authorisation field serves as a sanity check in the sense that it makes it impossible to attempt charges of more than \$500 at once. Thus, the online check only needs to be made for charges of \$500 or less.

Now, suppose K would like to order a game console priced at \$300 from the Internet. He has unfortunately forgotten that he has already used \$240 of his credit limit this month, so he will not have enough credit left. He writes the following certificate c_{KS} to the seller S.

c_{KS} = (cert (issuer K) (subject S) (propagate) (tag (may-charge \$300)) (not-after E) (online limit (uri svp:kv.net) K_v (once-only))) **(4)**

As can be seen from the certificate, it does not contain any information that would specify the "account" from which the charge may be made. If S were to possess a chain other than $\{c_{CH}, c_{HK}\}$ that would prove that K is authorised to make the transaction described in c_{KS} and to delegate that authority, then S might be able to get its \$300 from a different source. The use of a particular chain can be enforced through the use of different keys. If a particular key only has one authorisation, then there can be no confusion of which one to use.

K uses the validation server `kv.net` (with principal K_v) that ensures that authorised payments can only be charged once, and that K knows if the charge has been made or not. In practice, this validation server could be e.g. K's own terminal, which asks K to confirm.

K then acquires and sends the chain $\{c_{CH}, c_{HK}, c_{KS}\}$ to S. S then writes the certificate c_{HSval}, which will authorise C to check the validity of the certificates that require an online check.

c_{HSval} = (cert (issuer S) (subject C) (tag (validate hash({c_{CH},c_{HK},c_{KS}})) (nonce 666)) (not-after T+5min)) **(5)**

where T is the current time at the time of creating c_{HSval}. It should be noted that c_{HSval} only authorises the validation of certificates in the context of the specified certificate chain. This is to forbid another party (for instance, the credit card company) from constructing a different chain for the transaction, and using this authorisation for a purpose other than it was intended for.

Validation servers are naturally free to decide whose authorisations to trust, but in this example we follow the rule presented in Section 6.3. The validation server H_v only honors validation certificates issued by H, K or S. The validation server K_v only honors validity check authorisations issued by K or S. In general, H_v honors authorisations from those entities who appear in certificate chains after those certificates in which H_v is mentioned as the validation server; in this case, (possibly indirect) recipients of the certificate c_{HK} could all be accepted by H_v as a source of authority.

When S has the payment information and charge authorisation, it can make the charge if it has the product in stock and chooses to make the deal. It can do so by sending all of the certificates received from K together with the validation certificate that S itself wrote. C then makes the validity checks, and finds that the H_v replies that a check failed, because the charge attempted exceeds the limit set by H.

Had the limit not been exceeded, the online checks would have been successful, and C would then have committed to the transaction, and transferred the charged amount (minus any fee) to S's account. S should then deliver the ordered product to K.

The transaction must be handled in 5 minutes, or otherwise some of the certificates expire, which makes it impossible to complete the transaction.

9 Evaluation

According to the criteria introduced in Section 3, we can state that our protocol has the following properties:

Fail safety If the validation server fails to respond, the permissions should not be granted. This prevents denial of service attacks against the validation servers from hiding the fact that the certificate has been revoked.

Timeliness The validation protocol does not introduce any significant delay in the propagation of revocation information. Because everything is online, there is no need to use outdated copies of information. However, the notification and management of the validation servers may introduce some delay and is, therefore, a relevant topic for future work.

Adjustability The verifier can affect his own risk level by choosing to skip the online check based on the length of time elapsed since the same check was previously made.

Granularity The revocation can be performed on a per certificate basis, but not to individual permissions within a certificate. It should be noted that revoking certificates can affect third parties if the rights had been delegated.

Containment The validation server only controls the online validation and not the issuing of certificates. So, a compromise of the validation server will not facilitate the creation of new illegal certificates. The only exception is renew, where the validation server distributes new certificates. These certificates can, however, be issued offline in which case there is no problem with containment.

Reversal of revocation It is a simple matter of notifying the validation server that the revocation was an error or that the circumstances have changed and that the certificate should be re-enabled.

Protection of revocation This depends on the management of the validation server and is currently under work.

10 Future work

One way to improve the performance of long certificate chains is to use reduction certificates [6]. A certificate reduction certificate (CRC) replaces two or more certificates with one certificate so that this one certificate has the exact same meaning as the chain replaced. This reduction can be performed automatically and will make any future use faster. However, an unfortunate side-effect of the need for authorisation in limit validations is that it makes reduction over such certificates impossible. To verify the limit validation, we need an authorisation from the receiver of the original certificate or her descendant. However, if the receiver is removed from the chain by the reduction, there is no way of proving the descendence and, hence, the authority. This makes any further validations impossible and the CRC unusable.

Although certificate chain reduction certificates bring problems to the revocation protocol, they may be critical to the performance of the system. This would be the case especially in a widely deployed PKI with millions of certificates and potentially very long certificate chains. Thus, merely noting that chain reduction certificates cannot be created for chains that include online validity checks is not an attractive option in the long term. This is an issue that we are going to address in the future.

A possible other benefit of reduction certificates is the promotion of anonymity. However, if a reduction certificate contains online checks, anonymity might be compromised. Therefore, any online validation does not appear to be compatible with reduction certificates created for privacy purposes. If, on the other hand, the online validations can be performed before reduction and the resulting certificate has no online checks (though presumably a shorter validity period), the reduction might end up improving privacy.

Another issue that needs further attention is how the validation server finds out that the certificate is revoked. If the validation server is not the same server that issued the certificate or is otherwise responsible for making the revocation decision, an additional notification protocol may be needed.

The performance and scalability issues of certificate based systems in general and the validation protocol in particular still need further work. At the moment, they look promising, but without extensive empirical tests we can not state anything definite about their suitability as an Internet-wide solution.

In our project, we are also going to do further usability research on the subject of delegation management. So far we have built the underlying certificate functionality in a fairly technology-oriented manner, but the management issues really cannot be addressed without a strong emphasis on usability. In our usability research, we are trying to find out how certificates should be presented to users, i.e. how much must the users understand themselves and how much can be taken care of by the software. Furthermore, in a related research effort we are studying what makes users feel secure, i.e. which information the users want to see and what decisions they want to make themselves.

11 Conclusions

In this paper, we have discussed the different methods for certificate validation and revocation, and presented a protocol for authentication and certificate validation for SPKI based systems.

We conclude that certificate revocation lists are not the most attractive revocation method as they tend to transfer possibly large amounts of unnecessary information. We feel that online checks, which transfer only the relevant information and do not require storage of information that the party may never need, are more appropriate. As a consequence, we propose that

CRLs should be deprecated, if not removed and that the emphasis should be moved to online validations.

Using the authority delegated to a public key through a certificate chain requires a proof of possession of the corresponding private key. This is achieved using an authentication protocol. ISAKMP is a standard framework for key and security association agreement. We proposed to use the framework for certificate validity checks as well, and defined two new phase two exchanges for ISAKMP to implement our protocol.

We presented a set of design criteria a good protocol should fulfill and finished by analysing our protocol and concluding that we were able to satisfy most of them. The remaining ones were discussed and they are currently under work.

References

[1] Paul Ammann, Ravi S. Sandhu, and Gurpreet S. Suri. A distributed implementation of the extended schematic protection model. In *Proceedings of the seventh Annual Computer Security Application Conference*, pages 152–164, 1991.

[2] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform Resource Identifiers (URI): Generic syntax. *Request for Comments: 2396*, August 1998.

[3] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, California, May 1996. IEEECSP.

[4] Claudio Calvelli and Vijay Varadharajan. Representation of mental health application access policy in a monotonic model. In *Proceedings of 1993 IEEE Computer Security Applications Conf.*, December 1993.

[5] Carl M. Ellison, Bill Franz, Butler Lampson, Ronald L. Rivest, Brian M. Thomas, and Tatu Ylönen. Simple public key certificate. Internet draft (expired), IETF SPKI Working Group, March 1998.

[6] Carl M. Ellison, Bill Franz, Butler Lampson, Ronald L. Rivest, Brian M. Thomas, and Tatu Ylönen. SPKI certificate theory. *Request for Comments: 2693*, September 1999.

[7] Thomas Hardjono and Tadashi Ohta. Approaches to secure delegation in distributed systems. In *Proceedings of the 12th Annual International Phoenix Conference on Computers and Communications*, pages 188–194. IEEE Computer Society Press, March 1993.

[8] Dan Harkins and Dave Carrel. The Internet Key Exchange (IKE). *Request for Comments: 2409*, November 1998.

[9] I-Lung Kao and Randy Chow. An extended capabilities architecture to enforce dynamic access control policies. In *12th Annual Computer Security Applications Conference*, 1996.

[10] Hugo Krawczyk. SKEME: A versatile secure key exchange mechanism for Internet. In *Symposium on Network and Distributed Systems Security*, pages 114–127, San Diego, California, February 1996. Internet Society.

[11] Douglas Maughan, Mark Schertler, Mark Schneider, and Jeff Turner. Internet Security Association and Key Management Protocol (ISAKMP). *Request for Comments: 2408*, November 1998.

[12] Silvio Micali. Certificate revocation system. U.S. Patent 5666416. Issued September 9, 1997.

[13] Michael Myers, Rich Ankney, Rich Malpani, Slava Galperin, and Carlisle Adams. X.509 Internet public key infrastructure Online Certificate Status Protocol – OCSP. Internet draft, March 1999.

[14] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, Texas, January 1998. Usenix Association.

[15] Hilarie K. Orman. The Oakley key determination protocol. *Request for Comments: 2412*, November 1998.

[16] Ronald L. Rivest. Can we eliminate certificate revocation lists? In *Proceedings of the Second International Conference on Financial Cryptography*, Anguilla, British West Indies, February 1998.

[17] Ronald L. Rivest and Butler Lampson. SDSI – A simple distributed security infrastructure. (see SDSI web page at `http://theory.lcs.mit.edu/~cis/sdsi.html`).

[18] Ronald L. Rivest and Butler Lampson. SDSI – A simple distributed security infrastructure. In *Proceedings of the 1996 Usenix Security Symposium*, 1996.

[19] Ian Simpson. Modeling the risks and costs of digitally signed certificates in electronic commerce. In *Proceedings of the second USENIX Workshop on Electronic Commerce*, pages 287–297, Oakland, California, November 1996. USENIX.

[20] Stuart G. Stubblebine. Recent-secure authentication: Enforcing revocation in distributed systems. In *Proceedings 1995 IEEE Symposium on Research in Security and Privacy*, pages 224–234, Oakland, California, May 1995.

[21] Vijay Varadharajan and Claudio Calvelli. An access control model and its use in representing mental health application access policy. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):81–95, February 1996.

Evaluating Revocation Management in SPKI from a User's Point of View

Kristiina Karvonen[1, 2]***, Yki Kortesniemi***[1]***, Antti Latva-Koivisto***[1]
[1]*Helsinki University of Technology, P.O. Box 9800, 02015 HUT, Finland*
[2]*Visual Systems/ TietoEnator Ltd, Vanha Talvitie 19 a, 00580 Helsinki, Finland*
{Kristiina.Karvonen,Yki.Kortesniemi,Antti.Latva-Koivisto}@hut.fi

Abstract
The topic of computer and network security has gained an ever-increasing amount of interest in recent years. The pervasiveness of computers everywhere means that novel users, from novice to expert, need to be able to manage their own security in an understandable way, when giving information about themselves or making transactions online. In this paper, we will present, discuss, and analyse the various revocation methods of a Simple Public Key Infrastructure (SPKI) certificate based access control mechanism from a user's point-of-view. We will consider the downsides and benefits of each revocation method, and make recommendations for which methods to use in which use situations, and how to present the best choices to the user in an understandable way.

Key words: revocation, management, authorisation, certificates, usability

1. Introduction
Controlling access to valuable resources is a necessity, be they traditional resources such as a safe or a bank account, or computer resources, such as a file containing secret information. Managing the access can be a challenging task for the system designers and resource owners, particularly if the system is large and distributed. Currently, there are many kinds of security architectures used to manage the security of these systems. With most, the trend is to involve the users more in handling their security. The users have to decide whom to trust, to what extent, where and when.

Traditionally, access control systems have been based on the concept of Access Control List (ACL), where every resource is bundled with a list of authorised users. Typically, the list is located next to the resource, with all the relevant information in one place. An example could be the VIP list at the door of a club. If there are several doors, we need several copies of the list, and keeping them up to date requires extra work. Authorisation certificates reverse the concept, turning the centralised system into a distributed one. With certificate-based systems, the users of the resource are given a ticket that proves they have the right to use the system. The right no longer resides with the resource but with the user (all the VIPs have a special card they show at the door), and the right automatically follows the users to whichever copy of the resource they go to. If the club has several entrances, the VIP can use any one of them. A significant difference to the paper-based tickets is that authorisation certificates can be used to delegate the rights to other users without any help from the owner of the resource: users can delegate their own rights. This means that it becomes possible e.g. to create new credit cards that make it possible for children to use their parents credit right in such a way that the

parents keep their own card and the children have a limit to the amount they can charge from the card [Heikkilä-Laukka, 1999].

Certificates are intuitive in many ways – a certificate granted means a right granted – but sometimes there is a need to cancel, to revoke, a right previously granted. Unfortunately, the revocation mechanisms create problems from the user's point-of-view, because revoking a right is not as intuitive as granting a right, and because there are many mechanisms of revocation to choose from. The core of this paper is to pinpoint and answer the specific usability issues that these revocation mechanisms give rise to, and to be able to choose between them. We will do this by first having a look at existing research on usability of computer security – the area the work at hand falls into, describe the various revocation methods in SPKI, and outline the usability issues that these revocation methods give rise to with the help of several use cases. We will conclude with bringing together the outcome of the analyses of these use cases and their significance for the usability of revocation management in SPKI.

2. Usability of Security - Previous Work

Usability issues in revocation management are part of the problems in usability of computer security, still taking its first steps. It is a generally known fact that users are often considered to be the "weakest link", when computer security issues are at hand ([e.g., Adams and Sasse]). Rightly so, for no matter how sophisticated security mechanisms we use, they are only effective when used correctly. However, more and more "ordinary" people, without any former experience with security issues or technologies will have to learn to manage security now, and even more so in the future. One answer might be to increase the automation level of security. In this scenario, security would be taken care of by the system on behalf of the user, and the user need not bother about it. However, Whitten and Tygar [1999] state that even though automation may be the right solution for securing the communication channel itself, there remain situations where automation is not and cannot be the answer. At many points manual involvement is required from the user, for example when giving access to shared files for others. They also argue that usability of security has some specific usability problems not encountered in other areas. These include making users aware of the security tasks at hand, providing guidance throughout the procedure, and preventing dangerous errors [Whitten-Tygar, 1999]. Adams and Sasse [1999] give more or less the same recommendations for creating usable security in their treatment on how to make passwords user-friendlier, such as motivating users and providing feedback.

A further problem with usability of security is to define the right level of information provided. In case of certificate revocation, it seems we have two options: either we can hide the certificates from the users as fully as possible, or we must make the certificates understandable from the user's point-of-view. A mixture of hiding and revealing information about the certificates, along with preventing the user from making any serious mistakes, is a likely solution.

3. Revoking SPKI Authorisation Certificates

The SPKI authorisation certificates and their revocation methods have been developed by the Internet Engineering Task Force (IETF). The theory behind SPKI has reached the status of experimental RFC [Ellison & al, 1999], although the latest document has expired. The structure of the SPKI authorisation certificates has been derived from the theory document, but the document has not been completed and is therefore not an RFC. The revocation

mechanisms discussed in this paper are based on the latest draft of the structure document, but we have also included the proposed changes from Kortesniemi, Hasu and Särs [2000].

An SPKI authorisation certificate is essentially a ticket granting the specified right to the indicated recipient. The certificate is always valid and can be used an unlimited number of times, unless its validity is somehow limited by listing conditions in the validity field of the certificate. Once the resource owner issues a certificate, there is no practical method of getting it back from, say, a misbehaving recipient, so the issuer needs to include some limiting conditions in the validity field when the certificate is created. And here lies the difficulty: all possible future problems have to be anticipated and suitable countermeasures must be devised at the creation time. To better appreciate the problems involved, let us have a look at the various revocation methods available. Although there are six different methods to limit the validity of a certificate, only four (types C, D and E) can be considered revocation methods; the rest are just validity management methods. In this paper, we have grouped the methods in five types based on the speediness of revocation and expiration characteristics (Table 1). We can say that the types refer to the validity mechanisms themselves, or to certificates, whose most effective mechanism is of the type mentioned.

Table 1: The SPKI validity management methods

Type	Method	Speed of Revocation	Notes
A	No `Validity Period` / Only beginning time (= no end time)	N/A	Does NOT expire
B	End time / complete `Validity Period`	N/A	
C	`Renew`	After current certificate expires	
	`CRL`	After current `CRL` expires	
	`Reval`	After current "Bill of Health" expires	
D	`One-time`	Immediately	Can limit the usage of a group of users
E	`Limit`	Immediately	Can limit the usage of the particular user

The simplest method is a `validity period`: the certificate is valid only between the dates stated. However, both of the dates (not-before and not-after) are optional, so it is possible to create e.g. "eternal" certificates by omitting the expiration date (type A). Such certificates should be used only with careful consideration and are not likely to be seen by end-users (example: a computer granting the administrator all the rights to the computer). Once the validity period also contains an end time, we have a more regular certificate (type B). These kinds of certificates are good when the value of the right or the risks from misuse are not significant (example: one-day bus ticket).

When the value or the risk is high, we need some way of revoking the certificate. First, we look at type C that has three methods. `Renew` divides the long validity period into several shorter ones that are represented by individual certificates, and provides an automated method for fetching the subsequent certificate after the current one has expired. The issuer can at any time stop the distribution of new certificates so after the current (short lived) certificate expires, the right is revoked. `CRL` (Certificate Revocation List) is based on a periodically published list of revoked certificates. Revocation takes effect as soon as the subsequent list

has been published (i.e. after the current one expires). `Reval` is based on a periodically published "bill of health", which assures that a certificate is still valid. Without it, the certificate is invalid. Unlike `CRL`, `Reval` is not a list; it is issued individually to each certificate. These three methods appear similar to the issuer: the revocation takes place after a delay. This makes them suitable for a situation, where revocation is required, but the speed of revocation is not essential (example: one year bus ticket, which can be revoked every two weeks). The revocation methods of types D (one-time) and E (`Limit`) require contacting an online server every time the certificate is used. They can also be used to control the amount of use, not just to revoke the certificate completely. One-time can limit the usage on a general level (example: there are only 50 parking spaces in the garage, so limit the number of cars to 50), whereas `Limit` can actually control each individual certificate (example: this certificate allows 10 bus trips). The advantage of these methods is that revocation takes place immediately, but they also require a network connection.

4. Cases

In order to implement a system that is based on certificates, the user needs to have a terminal into which the certificates are loaded. Here, we consider a hand-held device type of terminal that has a screen and an ability to communicate with other systems either wireless or via a physical interface.

4.1 Bus Tickets

Helen, 15, wants to go shopping downtown, so she gets on a bus and buys a single ticket that is loaded onto her PDA. The ticket is valid for one hour, during which time she can transfer freely. In downtown, she needs to transfer to a tram to get to her favourite department store. This ticket scheme can be implemented with type B certificates. Helsinki City Transport (HCT) has delegated the right to sell tickets to all the buses and kiosks by issuing certificates to them. When Helen gets on the bus, the ticketing system of the bus creates a new certificate that is valid for one hour. The new certificate, together with the bus's certificate, is loaded onto Helen's PDA, which already contains a software module that can talk with the bus's system, and Helen's account is charged. When Helen transfers to the tram, the tram's system talks with Helen's PDA to check that she has the right to enter the tram. This ticketing scheme is rather simple. No revocation mechanism is needed, because the value of the ticket is very small. If Helen loses her PDA (and her ticket), the value of the ticket is her least concern. The issuer, HCT, does not have any interest in being able to cancel customer's tickets, because customers pay for their tickets in advance, and the customers cannot break any agreements that should result in the termination of the ticket.

Helen's father Matt wants to get to work in the morning and back to home in the evening, five days of a week, and an annual bus ticket is a good solution for him. He will not have to keep on buying new tickets, and an annual ticket will probably cost much less than, say, 500 single tickets. However, Matt is a forgetful person, and since an annual ticket costs around 600 €, he is worried about losing the PDA and the tickets with it. He needs some way to ensure that if the PDA does get lost, he will be able to revoke the ticket and get a refund for it. All this can be achieved with type B certificates, but for the user, losing the ticket involves too great a risk. In order to provide good and safe service for its customers, HCT should be prepared to issue new tickets to replace lost or stolen ones, so HCT should use a method where tickets can be cancelled. This requires the use of type C or type D methods.

With the type C methods, tickets cannot be cancelled immediately but only after a period of waiting. The `CRL` method requires that the list of cancelled tickets is updated onto every bus every morning, i.e. it requires a periodic connection to a central server. As a result, it may take up to 24 hours before a ticket becomes invalid. This system works well when there is only one issuer (HCT). It also requires that tickets cannot be resold or delegated, since that would mean more issuers. If there are many issuers, it gets impractical or impossible to obtain all the possible `CRL`s in advance, or alternatively, the required `CRL`s must be obtained online at the time when the ticket is used. In the simple case, the end-user's terminal is not required to go online, and the bus's system needs to go online only occasionally. The next method, `Renew`, is based on dividing the annual ticket into, say, two-week periods. At the end of each period, the ticket is renewed until one year has passed. In this case, we don't need a list of all cancelled tickets on every bus. Instead we get the problem of renewing the ticket. The ticket must contain the address of an online server where the renewed ticket can be obtained. This kind of renewal must not be the user's responsibility, since it does not match the user's goals. Let's consider the simple case. If Matt gets on the bus and the ticket is in the middle of a two-week period, everything goes well. The ticketing system on the bus checks Matt's ticket and lets him in. If one two-week period is coming to an end, Matt's ticket needs to be renewed. This can be done automatically by the user's terminal. The user's terminal should contact the online server and ask for a new ticket. When Matt gets on the bus next morning, everything is fine again, and Matt does not need to know anything about the renewal process. Alternatively, if Matt's terminal is unable to go online, the bus's ticketing system can renew the ticket when Matt gets onto the bus.

The third method, `Reval`, is based on revalidating the certificate again and again. It works very similarly to `Renew`, but here, the user's terminal must supply the ticketing system both with the certificate that is valid for one year and with a "bill of health" that is valid for two weeks at a time. Instead of renewing the certificate itself, the bill of health must be renewed, which makes the system more complex. From Matt's point of view, all the three different type C mechanisms are the same, so they should also look the same. The differences are technical and understanding them does not provide any added value for the end-user.

The problem of type C certificates is that they cannot be revoked immediately. This is both a usability and a utility problem. The user may have difficulties in understanding or accepting that his electronic ticket cannot be cancelled immediately. With type D certificates, where the certificate requires that its validity be checked from an online server each time the ticket is used, the ticket can, instead, be cancelled immediately. However, using type D certificates requires that the resource is always online when it is used. Alternatively, the user's terminal could be required to go online, but this would be poor service and poor system design. When we look this from Matt's point of view, we see that for Matt, the only thing that matters is that he does not have to bear the risk of losing his investment in the yearly ticket. This can be achieved with type C certificates by making a business decision. HCT could issue a new ticket for Matt if the old one is lost or stolen. Possibly HCT could charge a small fee for this service. HCT could then itself bear the risk that somebody uses Matt's ticket before it can be revoked. HCT could make this business decision, because this could be more economical than implementing a type D system, where all the buses must be online at all times. Also, the speed of the process is important for both Matt and HCT. If passengers must wait before they can enter the bus, it is irritating. If an actual implementation of type D system turned out to be too slow, a type C system should be used to ensure good quality service.

4.2 Parking House

Jane, Matt's wife, works for a company that rents 50 parking spaces for its employees from a nearby parking house, because it is known from experience that there are normally about 45 cars present at any one time, although there are almost 100 employees. The company then needs to distribute the parking permits to the employees. With the `One-time` method, we can implement a system for the garage that allows resource pooling and efficient use of parking lots. Every employee's PDA stores the certificate that grants access to the parking house. At the garage door, Jane's PDA automatically communicates with the door. The door lets Jane in only if less than 50 lots are in use. If all the rented spaces are occupied, the door tells Jane that all the spaces are full. For Jane, it is important to be able to know how many lots are still free. Therefore, the server that verifies the validity of the parking ticket (by checking the number of used lots) should also provide a facility to check the number of free lots. If Jane should decide to leave the company, her certificate can be immediately revoked, thus preventing any further use of the garage and without affecting any of the other employees.

4.3 Charge Card

Helen is about to leave for a two-month language course in London. During the course, Helen will need money for food, travelling, souvenirs and activities like theatre. Matt concludes that 300 € should suffice nicely and gives Helen access to his bank account for that amount. With a traditional charge card or a credit card, this would mean going to the bank and ordering a separate card for Helen (which can take several days). Further, such a card cannot be limited to any amount, so theoretically Helen could empty Matt's account.

When using certificates to implement a paying scheme, we can accomplish the limit [Heikkilä-Laukka]. Matt can go to his PC, open the bank program (used to pay bills) and create a "charge card" (certificate) for Helen, without any assistance or delay from the bank. These "charge cards" are stored in the user's PDAs and are used in shops to pay for products and services. For the certificate Matt needs Helen's "ID number" (public key), which is kept inside Helen's PDA. Helen gives the ID number by bringing the PDA next to the PC. Matt then keys in the limit (300 €), chooses the account from which the money should come, and finally sends the finished certificate to Helen's PDA. Technically, the limit of 300 € is implemented using the `Limit` method, which means that somewhere there is a server that monitors the usage of Helen's certificate. Theoretically, this could be any server, but in practice giving that choice to Matt would just make the system more difficult to use. Hence, it is quite natural that the server belongs to the bank, which Matt trusts already. The choice about which server to use must be made by the designer of the system, and Matt does not need to know the details.

During her course Helen can pay for her expenses by using the PDA at the cashier. The due amount appears on the PDA's screen, and once Helen accepts the sum, the money is transferred to the shop. Towards the end of Helen's course she finds out that a trip to Paris is being organised, but she no longer has enough money to participate. She decides to call her father and ask for a higher limit on her charge card. At that time, Matt is enjoying his summer vacation at the family cottage. Luckily, he has his PDA with him, so he can use it to raise the limit to £400 by simply typing a new value over the old one in the bank program. In Paris, the unexpected happens: Helen is pick-pocketed and she loses her PDA. She immediately calls her father, who can revoke the certificate at once.

In the case of a bank account/ charge card, only the `Limit` method can be used to control the amount of money spent on that certificate. If Matt had decided not to impose a limit, the bank's software would have chosen the `One-time` method instead, because the possibility of losing one's PDA still requires the possibility of immediate revocation. Implementing the system in this way is the responsibility of its designer. The system must not require that Matt make the decision about which kind of certificate to use. In order to make good decisions on the user's behalf, the designers must uncover the user's goals and needs before the system is implemented, and simulate their design from the end-user's point of view.

4.4 Summary of findings

Certificates can be used to enable various user tasks. In the cases just presented, the end-user does not need to be aware of certificates or revocation methods as such. They would be confusing by introducing new concepts that have nothing to do with the user's real goals. Instead, the end-user should be presented with concepts and information that match her goals: a bus ticket with an expiration time, a button to cancel someone's right to use one's bank account, or a phone number to call when the bus ticket or credit card is lost.

The designer of a system, however, has to understand certificates and revocation methods. It is the designer's responsibility to choose a suitable method with which to support the user's goals and to analyse which options are relevant to the end-user. To do this, the designer probably has to conduct field studies and observe and interview end-users. In particular, in each presented use case the required functionality could be reached with only one or two different revocation methods. Therefore, offering the end user a full list of methods is a bad idea. We also noted that `CRL`, `Reval` and `renew` can be made to look identical to the end-user (except in a limited set of cases), so the choice between them can be based on technical reasons.

The designer should follow established usability and interaction design guidelines in creating the system. Especially visibility of the user's data and the system's state are important. The end-user must be able to see what rights she has acquired, such as a bus ticket. The validity periods and limits for such rights should be shown. With the `Limit` method (type E), it is important that the server is able to tell the remaining limit, in addition to validation checking. The user must be able to easily see, to whom she has delegated rights, and revoked permissions must also be clearly visible. When presenting user's data and the system's state, all information should be shown in its context. The delegated right to use one's account must be presented together with the account in the bank application.

The two reasons for revocation – the issuer discovers that the receiver is misusing the certificate and wants to take it away, or the receiver loses the certificate and wants it replaced, in which case the old one has to be revoked – enable us to make the following conclusion on the different types. Type A certificates should not be offered as an option to end-users, because they are valid forever and cannot be revoked. Type B certificates are a good choice, when the value of the right is not significant and the end-user cannot cause significant damage to the issuer by misusing the certificate. Type C certificates could come into play when the value becomes so big that a revocation method is necessary, but the misuse does not require immediate revocation – for instance, when misuse is rare and the issuer can understand and bear the risk of misuse. This is how credit card companies handle misuse today. However, it must be noted that for the end-users it is important that they do not need to bear the risk. The system should be such that from their point of view, the revocation takes place immediately.

Finally, the online methods should be used when immediate revocation is desirable. The choice then depends on whether we want to limit individual certificates (type E) or just the certificate group (type D).

An implementation issue with type C is the required online connection. If the validity information is fetched by the user's PDA, it should take place automatically so that the operation is transparent to the user. If the user has to take some steps to get the information, it makes the system appear much more complicated and means unnecessary work. Since currently many PDAs are not capable of online connections, it seems that the resource, e.g. the bus, should fetch the validity information. In this case, however, the resource has to be online all the time, because it usually cannot anticipate the required information, whereas the PDA only requires occasional connections. Only in limited situations where there are a very small number of `CRL` issuers (the HCT in the bus ticket example), can the `CRL`s be regularly fetched, and hence the connection is not required continuously. Another implementation issue is the feedback time. If the response time of a higher level revocation method (type D or E) is too long, say, over a couple of seconds in the bus example, the designer should choose a lower level method (type B or C), since immediate feedback and fast operation are important for the end-user.

5. Conclusions

On basis of the above analysis, we can clearly see that from the end-user's point of view, the revocation methods indeed have differences. For example, the types D (`One-time`) and E (`Limit`) can be recommended, since they provide the users with the greatest amount of control over the revocation management and are relatively easy to understand. However, we have also seen that the usability issues within revocation management are manifold and cannot be resolved without a thorough understanding of the specific use situations the system is designed for, nor without knowing who will be using these systems, where, when and how.

References

Adams, A, Sasse, M.A: Users Are not the Enemy. Communications of the ACM, December 1999, Vol. 42, No. 12, pp.41-46.

Ellison, C, Franz, B, Lampson, B, Rivest, R, Thomas, B and Ylönen, T. SPKI certificate theory. Request for Comments: 2693, September 1999.

Heikkilä, J, Laukka, M: SPKI based Solution to Anonymous Payment and Transaction Authorization, Proceedings of the Fourth Nordic Workshop on Secure IT Systems (Nordsec'99), 1999, Kista, Sweden

Kortesniemi, Y, Hasu, T and Särs, J: A Revocation, Validation and Authentication Protocol for SPKI Based Delegation Systems, Proceedings of Network and Distributed System Security Symposium (NDSS 2000), 2-4 February 2000, San Diego, California

Whitten, A. and Tygar J.D: Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. Proceedings of the 8th USENIX Security Symposium, August 1999.

Validity Management in SPKI

Yki Kortesniemi
Helsinki University of Technology
Yki.Kortesniemi@hut.fi

ABSTRACT

In a distributed system, using authorisation certificate based access control tends to facilitate the granting of rights. On the other hand, the problems of limiting usage or revoking the rights become more difficult, as the issuer of the right is no longer in control of the issued certificate.

In this paper we take a look at the role of certificates in access control, evaluate the technical merits of different validity management mechanisms an SPKI authorisation certificate supports, discuss the problems related to managing the validity and finally introduce a protocol for validity management.

1. Introduction

Access control becomes an interesting question whenever an entity controls some resource that others would like to use. In the absence of control, a resource likely ends up being exploited without any benefit to the owner. A computer related example is the protection of a database system. Traditionally, this has been implemented using an ACL (Access Control List), which lists the authorised usernames and their associated rights. This solution has many good qualities in mainframe-type systems, but in a distributed environment with multiple instances of the database, problems arise because we are relying on a central list. Solutions like replication can be used to lessen the impact, but essentially an ACL is a centralised solution.

Authorisation certificates, on the other hand, yield themselves quite naturally to a distributed environment. SPKI certificates, for instance, can successfully be used to implement systems that support anonymity, delegation and dynamic distributed policy management – all qualities not traditionally associated with ACLs. The key idea in authorisation certificates is to give the user an unforgeable ticket, which states the user's rights, thus making ACLs unnecessary. The verifier monitoring the resource simply has to make sure that the certificate is valid, originates from the verifier and has been granted to the user in question, before giving the user access to the resource. It is interesting to note that Kerberos combines elements from both ends: it maintains the long term access information in the server's database (ACL), but the actual access control decisions are based on short-lived tickets not unlike certificates. However, actual authorisation certificates tend to be much longer-lived and do not normally rely on a backing ACL.

The self-containment is a strong point of authorisation certificates, but also the source of one of their weaknesses: the difficulty of revoking them. With ACLs, revocation is easy: just erase the unwanted entries. With certificates, the problem is more complicated, because instead of the issuer, the user is in control of the certificate. Therefore, all the revocation solutions for SPKI certificates rely on additional online checks. By using online servers, we lose the self-containment, but this loss is often acceptable. Nevertheless, using these revocation mechanisms always has a performance impact on the system, and they should therefore be used with consideration.

The immutability of certificates, unfortunately, also makes it difficult to keep track of the amount of usage – we cannot just cross out a part of the certificate as a sign of usage, we need other methods. One solution proposed in [10] is to use online servers to keep track of usage, thus enabling the use of tickets that are valid 10 times or credit cards that have monthly limits. However, managing this limit presents some problems.

In this paper, we take a look at the validity management options of one particular authorisation certificate, namely Simple Public Key Infrastructure (SPKI) certificate[7][8], study the problems of managing them and finally offer a solution in the form of a revocation management protocol.

The intended application domains could include things like organisations, which want to control their internal access rights – in these cases the users identity is usually known by the administrators granting the rights and the user might have several rights assigned to the same public key. At the other end we have global applications, where consumers buy some access rights with cash (e.g. the right to read the current issue of a particular magazine) and want to stay anonymous. In this case, the user might create a new public key for every right bought just to enhance privacy.

The rest of the paper is organised as follows: we first look at access control and how certificates can be used for it. Then, we look at SPKI certificates and their validity management methods, discuss their suitability for different situations and finally present a protocol for managing the online servers.

2. Access Control and certificates

The goal of access control is to make sure that only authorised users (be they humans or computers) get access to the protected resources. The access control process therefore can be said to consist of the following phase (depicted in Figure 1):

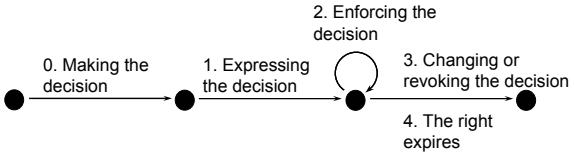

Figure 1. Phases of access control.

0. Making the decision
   In this phase, the issuer (someone either owning the resource or having the right to control access to it) makes the decision to grant a subject the right to access the resource. This decision could be based on things like the issuer knowing the subject (a friend), the subject holding some position in issuer's organisation or the subject being a paying customer to issuer's service.

1. Expressing the decision
   For the decision to be automatically enforced, it has to be expressed in a precise format. This could be e.g. an ACL entry or an authorisation certificate.

2. Enforcing the decision
   Whenever the subject tries to use the resources, the validator makes sure that the right still exists. This could entail checking the subject's credentials or the ACL and verifying that the subject is indeed the same as mentioned in the credentials or in the ACL.

3. Changing or revoking the decision
   Should the access right become insufficient, unnecessary or should there be risk of misuse, the right can be changed or even revoked.

4. The right expires
   Eventually, the right expires, either intentionally, or implicitly.

2.1. Different types of certificates

There exist three major types of certificates: identity certificates (e.g. X.509 and PGP), authorisation certificates (e.g. SPKI) and attribute certificates as shown in.

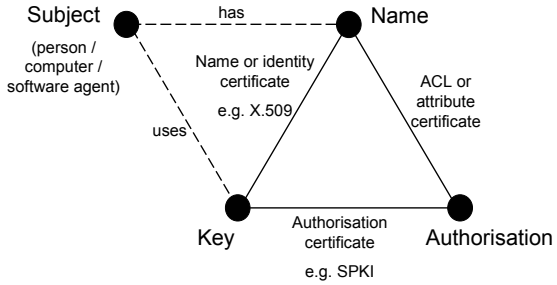

Figure 2. Three major types of certificates.

An identity certificate binds a public key to a name so that outside parties can be convinced that a particular person uses a particular key. Of course, this entails that the issuer (typically an organization called Certification Authority, CA) actually makes sure that the key is controlled by said person. Hence, CAs must be trusted by all users and they tend to be large organisations.

An authorisation certificate, on the other hand, binds a right to a public key. Authorisation certificates can be issued by anyone owning a resource or having the right to grant access to someone else's resource. This means that potentially every human, computer, or even a software agent could be issuing certificates. This difference in the number and resources of issuers between the two certificate types has significant implications on the revocation systems used, as we'll later discuss.

The third, a less common type, an attribute certificate, is used to bind an authorisation to a name. Essentially, it is a distributed version of an ACL.

To better appreciate the differences between identity and authorisation certificates, let us briefly look at how they are utilised in phases 1 and 2 of the access control process. In phase 0, certificates play no role, and the role of authorisation certificates in phases 3 and 4 is the subject for the rest of the paper. In this discussion, we assume the usage of public key based authentication.

2.2. Using certificates in phase 2: Enforcing the decision

To fulfil phase 2 in the access control process, we have to prove the binding between the subject requesting access and the required right. As we can see from Figure 2, there are several ways of doing this. In all of these, the binding between the subject and the key is assumed much tighter than the binding with password. This assumption however does not always hold, as the subject can either lose the control or just give the required private key away. In both these situation, revocation of that key and the associated rights is normally required.

The most common way of using certificates is to use identity certificates to establish a mapping from the key to a name and then use ACLs or attribute certificates to map the name to an authorisation. This approach nicely extends existing solutions, but it also has many problems:

- By design, it makes anonymous usage impossible. In some system, it is a requirement to prevent anonymous usage, but in other cases knowing the user's identity is not a necessity; it merely promotes unnecessary monitoring of users.

- Making a tight binding through the name is not easy, as it requires names that unique within the application domain – otherwise namesakes can share their rights. If we have a small organisation, this might be quite feasible, but even in a moderately sized organisation there can be more than one John Smith and we have to be very careful never to mix them up. And if we make global consumer applications, we need globally unique names, which are difficult for humans and impractical for computers. The local names can be complemented with additional information to make them global, but for global applications it is more straightforward to use global identifiers like public keys from the beginning.

- The binding from a key to an authorisation is unnecessarily long – it consists of two steps: key to name and name to authorisation. This is an important aspect, as the verification of this binding will be performed many times – in fact, every time the subject uses the resource.

However, this two step binding does present an advantage: by revoking the identity certificate we can automatically revoke all the associated rights (naturally, this is an advantage only if there are several rights associated to a single certificate). Further, we can create a similar construct with authorisation certificates, if necessary, so this is not a unique advantage of identity certificates.

An authorisation certificate, on the other hand, makes a direct binding from the key to the authorisation. This makes the binding simpler, but also practically anonymous. In reality, the key is a pseudonym, but since these pseudonyms do not have to be registered anywhere, it can be very difficult to trace them back to the user's identity. And, should the anonymity become a problem, it can be circumvented by verifying the subject's identity already in phase 1 (but if this is omitted, we cannot perform it retroactively).

Based on the above, we can conclude that authorisation certificates offer a simpler solution to phase 2 than solutions based on identity certificates.

2.3. Using certificates in phase 1: Expressing the decision

This phase is a more natural application area for identity certificates. They are often used to get the unique name of the subject, which is then used in the ACL or in an attribute certificate. But as we saw, this approach presents some problems.

Another way of using identity certificates is to acquire the known user's public key to issue them an authorisation certificate. This applies to situations such as issuing rights to members of an organisation. It should be noted, however, that identity certificates are not always necessary for issuing authorisation certificates. For instance, we could receive the public key directly from the subject in a face-to-face meeting, in which case an identity certificate is unnecessary.

2.4. Additional advantage of authorisation certificates - delegation

If the certificate does not expressly forbid it, it is possible to delegate the rights listed in the certificate to other users without any help from the owner of the resource - a feature, which makes distributed management easier to organise than in centralised solutions. In fact, regular users can delegate their own rights. For example, this means that we can implement a scheme, where a parent can issue a copy of her credit card to a child and limit the amount the child can charge from the card, while still keeping her own credit card [9].

The downside of this flexibility is that the certificate chains can become very long and evaluating them is no longer trivial. The solution is to view the chains as a means of implementing the granting of rights and then let the verifier automatically create a reduction certificate that replaces the chain with a single certificate, thus making the usage of the right efficient.

3. The SPKI Certificates

The Internet Engineering Task Force (IETF) has been developing SPKI as a more flexible alternative to X.509. The key idea is that anyone (or anything) with access to a resource can authorise others to use the resource by issuing them an authorisation certificate. So, compared to X.509, where only CAs issue certificates, in SPKI any person, computer, etc. can issue certificates - and also has to be able to manage their validity.

Altogether there are six validity options in SPKI certificates. The simplest and the only locally evaluateable is the validity period. In addition, the current SPKI structure includes three online validity checks: CRLs, revalidations and one-time checks. Furthermore, [10] proposes formats for two additional online validity checks: limit and renew. As we shall later see, the different methods can be ordered by increasing capability. Therefore, using more than one online method in the same certificate is usually redundant since the most capable suffices (although the selected method can be used several times).

The author's model for the lifecycle of an SPKI certificate is depicted in Figure 3. Each new certificate begins its life in the suspended state (transition 1), but the certificate moves to the available state when its validity period, crl and reval permit, possibly even immediately (transition 2). In the available state, the certificate can be used, provided that one-time and limit agree (transition 3). Should the crl or reval methods be used to revoke the certificate, it moves to the suspend state if it can later become available again (transition 4), or to the expired state if it no longer can be made available (transition 5 and 6). Finally, the certificate should naturally expire as dictated by the validity period (transitions 7 and 8). The renew method (transition 9) complements the model by forming a chain of shorter lived certificates – once a short-lived certificate expires, the subsequent one is ready to take its place (though it could be argued that the validity periods of consecutive certificates might be allowed to overlap).

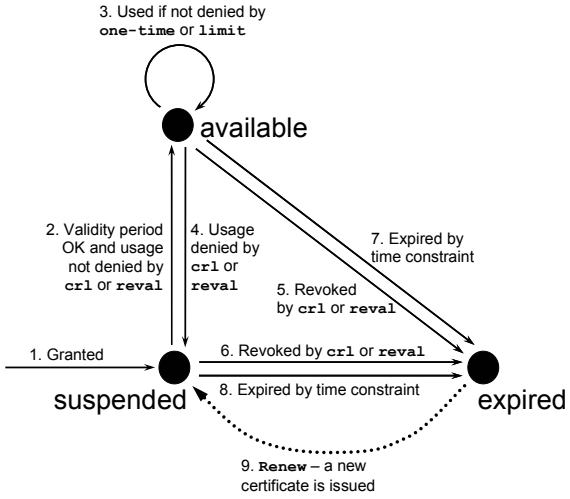

Figure 3. The lifecycle of an SPKI certificate.

3.1. Validity periods

In SPKI, the validity period definition consists of two parts:

```
<not-before>::
    "(" "not-before" <date> ")" ;

<not-after>::
    "(" "not-after" <date> ")" ;
```

Both parts are optional, and if either one is missing, the certificate is assumed to be valid for all time in that direction.

```
<valid-basic>::
    <valid-date> | <valid-dates> ;

<valid-date>::
    <not-before> | <not-after> ;

<valid-dates>::
    <not-before> <not-after> ;
```

There is an additional type of validity period called ``now", which has a length of 0, and can only be the result of an online check. It is interpreted to mean that the certificate is valid the moment the validation request was made, but it states nothing about the future. If the same certificate is used repeatedly, the online check has to be repeated, as well.

To facilitate the decision of whether or not a certificate is valid at a particular instance of time, all the different validity conditions end up being converted to validity periods as specified above. So, validating a certificate is relatively straightforward: check that the validity period stated in the certificate, as well as the online checks (converted to validity periods), are all valid at the time of use, and the certificate as a whole is then valid, and, therefore, grants the included permission.

3.2. Online checks

All the online checks are defined using the following format:

```
<online-test>::"(" "online"
    <online-type> <uris> <principal>
    <s-part>* ")" ;
```

where `<online-type>` can be `crl`, `reval`, `one-time` or `limit`. The `<uris>` specify one or more URIs (Uniform Resource Identifier [6]) that can be used to request revalidation; e.g. in the case of `crl`, the URI points to the crl file. `<principal>` specifies the public key used for verifying the signature on the online reply. The `<s-part>` is optional, and may contain parameters to be used in the online check.

Next, we'll go over the individual methods and their reply formats.

3.3. CRL

CRL (Certificate Revocation List) is based on the idea that a certificate is valid unless it appears on the specified CRL. SPKI includes both traditional and delta CRLs in its specification. These must also be signed by the aforementioned `principal`. The CRL formats are specified below.

```
<crl>::"(" "crl" <version>?
    "(" "canceled" <hash>* ")"
    <valid-basic>")" ;

<delta-crl>::"(" "delta-crl" <ver-
    sion>? <hash-of-crl>
    "(" "canceled" <hash>* ")"
    <valid-basic> ")" ;
```

3.4. Reval

Reval is based on an opposite idea: the certificate is invalid unless the prover can provide a current ``bill of health", which testifies that the certificate can be considered valid for the stated period. [10] specifies the reply format:

```
<reval-reply>::"(" "reval"
    <version>? "(" "cert" <hash> ")"
    "invalid"? <valid-basic> ")" ;
```

The reply identifies the original certificate in the hash and gives a confirmed (in)validity period for that certificate. The reply must be signed with the key given as `<principal>` in the original certificate.

3.5. One-time

One-time is based on the idea that it is impossible for the issuer to predict anything about the future validity of a certificate and, therefore, the user has to check the validity with every use of the certificate. The certificate contains a URI to the server, and the reply is ``yes" or ``no" with a time period of ``now".

```
<one-time-reply>::"(" "one-time"
    <version>? "(" "cert" <hash> ")"
    "invalid"? "(" "one-time" <nonce>
    ")" ")" ;
```

Again, the hash must correspond to the original certificate, and the reply message must be signed by the principal given in the certificate.

3.6. Limit

Limit is meant to enable quotas, i.e. it can be used to limit the usage based on suitable properties, like the number or length of usage. It is otherwise similar to one-time except that the server will not reply to queries, unless the user is able to prove that she is authorised to use the resource in question by presenting a suitable certificate chain. The limit query sent to the online server is of the form:

```
<limit-query>::"(" "test" <version>?
    "limit" <cert> <request>? <chain>
    ")" ;
```

```
<request>:: "(" "request" <s-part>
    ")" ;

<chain>::"(" "chain" <cert>+ ")" ;
```

Above, <cert> is the certificate whose online test(s) are to be made, <request> specifies the amount of resources requested, and <chain> proves that the verifier is entitled to ask about the validity of the certificate. The last certificate of the chain must be the validation certificate, which contains the <nonce> that is to be included in the reply to the query.

```
<limit-reply>:: "(" "limit"
    <version>? "(" "cert" <hash> ")"
    "invalid"? "(" "one-time" <nonce>
    ")" <context> ")" ;

<context>:: "(" "context" <hash> ")"
    ;
```

where <hash> is a hash of the concatenation of the canonical forms of <request> and <chain>.

3.7. Renew

Renew offers an alternative approach to revocation. Instead of issuing long-lived certificates and then worrying about their validity, we issue a string of short-lived certificates, which together cover the lifetime of a long-lived certificate. This simplifies matters, as the short-lived certificates can often operate offline and the network connection is required only to automatically fetch the subsequent certificate.

If everything is in order, the reply contains the next certificate:

```
<renew-reply>:: "(" "renew" <ver-
    sion>?  <cert> ")" ;
```

If, however, the right has been cancelled, the reply is of the form:

```
<renew-reply>:: "(" "renew" <ver-
    sion>? "(" "cert" <hash> ")"
    <valid-basic>? ")" ;
```

Again, the hash must correspond to the original certificate and the validity period states a period of time during which renewal requests will be denied (i.e. the conceptual long-lived certificate is not valid during this period).

4. Related work

The majority of work done in the field of certificate revocation has so far concentrated on identity certificates, in particular X.509 identity certificates. There exist several RFCs and Internet drafts that deal with X.509 certificate management and validation [5][1][2][3][4][14][12]. As to revocation methods, most of them concentrate on the CRL concept, and on how to effectively use it, but lately the trend has been to introduce other methods including online methods.

As to research, the majority of work has concentrated on evaluating the efficiency of CRLs and implementing improved, yet similar solutions. Further, some different solutions have been proposed [13]. Some work has also concentrated on the risk models and on the evaluation of different mechanisms in light of these risks [15][11]. Unfortunately, compared to SPKI authorisation certificates, there are a few significant differences in the X.509 model, which prevent us from directly applying the same solutions:

- The number of certificate issuers. In X.509, the number of CAs that issue certificates is orders of magnitude smaller (in SPKI, every human, computer etc. can issue certificates). This makes CRLs, which aggregate revocation information, much more feasible.

- Risk model. In X.509, the issuer and verifier are normally separate entities. The risk is taken by the verifier, yet the revocation decisions are made by the issuer. In SPKI, the risk takers are also issuing the certificates and can therefore control the revocation decisions to balance the risk.

These issues have been discussed in more detail in [10]

5. Choosing the validation and revocation methods

The phases of access control were presented in Figure 1. In [10] we have discussed the revocation problems at the time the certificates are used (phase 2 in Figure 1). These include the problems of authenticating the participants and providing undeniable evidence, also for liability reasons. In this paper, we focus on phases 1, 3 and 4. In phase 1, the essential problems include choosing the right validation methods, choosing the servers to implement them, informing the servers about the validity rules, and possibly paying the server's owner, if the servers are operated by a third party. In phase 4, the

Table 1: A summary of the online methods

Method	Typical use	Processing overhead	Revocation speed
Limit	Quota	High	Immediate
One-time	Limit usage on non-user specific factors	Moderate	Immediate
Reval	Revocation	Low	After current reval validity period
CRL	Revocation	Low	After current crl validity period
Renew	Revocation	Low	After current certificate expires

problems include things like informing the server about the revocation decision and providing undeniable proof, again for liability reasons.

5.1. Validity period

Phase 4 is simply a mechanism for making sure that certificates do not remain valid indefinitely, but instead automatically expire after a reasonable time. As a rule, it is a good practice to always include an expiration date in a certificate (only in very rare situation are there good reasons to make it a permanent certificate). In most of the cases, the matters themselves tend to change over time, so it makes sense to periodically reissue the certificates, if the rights are still required. Otherwise, the issuer is stuck with a growing number of certificates, which cannot be purged from the systems, as they are still officially valid.

5.2. Choosing the online method

This section discusses some of the main criteria in choosing the most suitable revocation method for a particular situation. Most, if not all, of these choices should be made by the designer of the system - they should not be left to the end users. [9] provides further examples of cases for each method and how they affect the end user. The results of this discussion have been summarised in Table 1.

An authorisation certificate is essentially a ticket granting the specified right to the indicated recipient. Now, the certificate is always valid unless its validity is somehow limited by listing conditions in the validity field of the certificate. Once the certificate has been issued, there is no practical method of getting it back from, say, a misbehaving user. The only recourse the issuer has is to include some limiting conditions in the validity field when the certificate is created. Here lies the difficulty: all possible future problems have to be anticipated and suitable countermeasures devised at the creation time. This is almost a mission impossible, because delegation will take place - the final user cannot be known until the time the certificate is used.

The choice of the most suitable validation/revocation method depends on what we want to achieve with it. We typically have two different goals: to control the amount of usage either discriminately (limit) or non-discriminately (one-time), or just to enable the revocation of the right in case the circumstances change, there is misuse of the right, etc. With the proposed changes to SPKI, any of the online methods can be used for the latter.

In the latter use, one important aspect is how fast we want our revocation command to take effect. CRLs and reval are both issued with a validity period, which is then the worst case time the issuer has to wait for her command to take effect. On the other hand, making the period very short does have performance implications, as the users are then forced to be online more often and fetch the latest validity statement. The issuer can naturally vary the validity period depending on the rate of problems or some other factor, but essentially both methods are best suited for situations, where the validity period does not have to be very short. This is particularly true about CRLs, where the validity period has to be the same for all certificates on the list, thus making it less practical to shorten the period if one of the certificates is showing signs of misuse. Processing overhead for the online server is fairly low with both of these methods, as the same reply can be used throughout the validity period.

On the other hand, a typical end user, e.g. someone using a certificate-based credit card, is less interested in the performance problems and more interested in the system behaving in an intuitive manner: when the parent presses the button to revoke the child's credit card, the revocation should take effect immediately, not after some arbitrary time. Even if security-wise this time might not be that important, compared to the time it might have taken for the parent to realise that security has been breached and that the certificate should be revoked, the delay is still a source of anxiety to the parent and should whence be minimised. For that reason, a method like CRL or reval is not good: they sacrifice the sense of control for the benefit of reduced overhead.

The only additional advantage they offer is support for offline operation, which is not necessary in all situations. On the other hand, the delay does not have to matter to the end user – the possible misuse and its costs can be included in the business model of the system, similarly to the existing credit card systems [9].

One-time is more suitable in a situation where we essentially want our revocation decision to take effect immediately or at least with a very short delay. On the other hand, we pay a price in performance for this convenience – every instance of usage requires network connection, as well as an individual reply from the server. So, if the certificate is used very often and performance really becomes a problem, we might consider using a lighter method and taking care of the misuse with the business model as mentioned above.

The other use for one-time, namely, controlling the amount of use, is another matter. In this case, we consider the certificate to be a recommendation, but the actual right depends on the circumstances, like the time of day or current load on the system. In this case, we are most likely more than willing to accept the performance penalty in exchange for the additional functionality.

Finally, limit is most likely used for controlling a quota; the possibility of revocation is just a fringe benefit. In this case, we pay an even higher price in performance, as its usage requires a two-phase negotiation with individual replies, but the new possibilities should more than outweigh that.

6. Background for the validity management protocol

In this section, we go over some of the key questions in designing the protocol.

6.1. Who can issue commands?

One of the basic things is naming the principal(s) that are allowed to issue revocation commands. The most obvious solution would be to state that the principal, who issued the certificate, is implicitly assumed to have the right to revoke it. However sometimes it would make sense to authorise others to revoke a particular certificate, for instance in a situation, where it is imperative that the certificate is revoked as fast as possible after a breach but the original issuer is not available to perform the revocation.

6.2. Requesting status information

The issuer might be interested in following how the certificate is used, particularly if it contains one-time or limit conditions, or if there are several individuals with the ability to revoke the certificate.

6.3. Auditability

The commands and their replies have to be auditable in case there is dispute as to the correct replies given by the server.

6.4. Support pre-evaluated answers and dynamic answers

In some cases, the answers are known in advance, e.g. when we revoke a certificate. In other situation, like with one-time and limit, we want to evaluate the answer at the time of usage.

7. SPKI Validity Management protocol

The protocol has been defined in XML and corresponding DTD can be found in appendix A. It defines the structure and contents of the messages between the issuer and online server. All messages are signed, which guarantees message integrity and authentication. Further, to protect against replay attacks and to guarantee confidentiality, a secure transport layer is used to carry the messages.

The protocol consists of just two messages: a command and a corresponding reply.

7.1. The Command

The command has the following structure:

```
Server_update cert, chain?,
    online_test_hash, de-
    lete_request*, test_definition*,
    status_query*, signature
```

`Cert` is the certificate, whose online condition is being managed. `Chain` is an optional field containing a list of certificates that proves that the current command issuer is authorised to send the command (this is required only if the command is sent by someone other than the certificate issuer). `Online_test_hash` identifies, which one of the possibly multiple validity conditions in the certificate is being managed.

The following three fields form the main part of the message. Even though they all are optional, at least one of them must be included in the command for it to be valid. The first, `delete_request`, defines which already defined rules are to be deleted. Each `delete_request` contains a validity period; all rules applying to that validity period are to be deleted.

The next part, `test_definition`, issues the new validity rules. There are two types of rules: pre-evaluated answer to be distributed at the specified time, and dynamic code that is to be evaluated by the server when a request is made. The pre-evaluated answer is further divided in three classes: a yes_no_answer is used for reval and crl, i.e. methods that reply with a validity period, Now_answer is used for one_time and new_cert_answer is used with renew. Limit always requires a dynamic_condition.

The final part, `status_query`, requests information on the validity status. It defines validity period for which we want the status information. Further, with the verbose flag the server is instructed to include in the reply the rule used to deduce the status.

The command ends with a `signature`.

7.2. The Reply

The reply follows a similar structure:

```
server_reply cert_hash,
    online_test_hash, delete_reply*,
    test_definition_reply*,
    status_reply*,service_status,
    signature
```

`Cert_hash` is a hash of the certificate in question. `Delete_reply` and `test_definition_reply` contain status codes about the success of the corresponding commands. Finally, `status_reply` contains status information for the requested periods and optionally the rules for deducing those.

8. Conclusions

In this paper, we have discussed the problems of managing the online validation and revocation of SPKI authorisation certificates. Due to their nature, authorisation certificates are well suited for granting rights, but limiting or revoking them presents a bigger challenge.

All the existing solutions to these problems are based on online servers that give authoritative statements about the validity of a certificate. We have discussed the advantages and drawbacks of the various solutions. Finally, we have presented a protocol for managing the online servers.

9. References

[1] C. Adams, P. Sylvester, M. Zolotarev, R. Zuccherato: Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols. Request for Comments: 3029, February 2001.

[2] C. Adams, S. Farrell: Internet X.509 Public Key Infrastructure Certificate Management Protocols. Request for Comments: 2510, March 1999.

[3] C. Adams, S. Farrell: Internet X.509 Public Key Infrastructure Certificate Management Protocols. Internet Draft, December 2001.

[4] Ambarish Malpani, Russ Housley, Trevor Freeman: Simple Certificate Validation Protocol (SCVP). Internet Draft, March 2002.

[5] A. Aresenault, S. Turner: Internet X.509 Public Key Infrastructure: Roadmap. Internet Draft, January 2002.

[6] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform Resource Identi_ers (URI): Generic syntax. Request for Comments: 2396, August 1998.

[7] Carl M. Ellison, Bill Franz, Butler Lampson, Ronald L. Rivest, Brian M. Thomas, and Tatu Ylönen. Simple public key certificate. Internet draft (expired), IETF SPKI Working Group, March 1998.

[8] Carl M. Ellison, Bill Franz, Butler Lampson, Ronald L. Rivest, Brian M. Thomas, and Tatu Ylönen. SPKI certificate theory. Request for Comments: 2693, September 1999.

[9] Kristiina Karvonen, Yki Kortesniemi, Antti Latva-Koivisto. Evaluating Revocation Management in SPKI from a User's Point of View, Proceedings of Human Factors in Telecommunication 2001, November 2001, Bergen, Norway

[10] Yki Kortesniemi, Tero Hasu, Jonna Särs: A Revocation, Validation and Authentication Protocol for SPKI Based Delegation Systems, Proceedings of Network and Distributed System Security Symposium (NDSS 2000), 2-4 February 2000, San Diego, California

[11] Patric McDaniel and Aviel Rubin. A Response to "Can We Eliminate Certificate Revocation lists". In Proceedins on the Financial Cryptography '00. The International Financial Cryptography Association (IFCA)., February 2000.

[12] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. Request for Comments: 2560, June 1999.

[13] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. In Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas, January 1998. Usenix Association.

[14] Denis Pinkas, Russ Housley: Delegated Path Validation and Delegated Path Discovery Protocol Requirements (DPV&DPD-REQ). Internet Draft, April 2002.

[15] Ronald L. Rivest. Can we eliminate certificate revocation lists? In Proceedings of the Second International Conference on Financial Cryptography, Anguilla, British West Indies, February 1998.

Appendix A: The DTD of SPKI Validity Management Protocol

```
<!--
   DTD for a SPKI online test management messages.
-->
<!ELEMENT hash                  EMPTY>
<!ATTLIST hash          data CDATA #REQUIRED>
<!ELEMENT cert_hash     hash>
<!ELEMENT cert          EMPTY>
<!ATTLIST cert          data CDATA #REQUIRED>
<!ELEMENT chain                 (cert+)>
<!ELEMENT online_test_hash      hash>
<!ELEMENT reason                (#PCDATA)>
<!ELEMENT no            EMPTY>
<!ELEMENT notbefore     (#PCDATA)>
<!ELEMENT notafter      (#PCDATA)>
<!ELEMENT date                  (#PCDATA)>
<!ELEMENT valid                 (notbefore?, notafter?)>

<!ELEMENT yes_no_answer no?, valid>
<!ELEMENT now_answer  no?, valid>
<!ELEMENT new_cert_answer       cert, notbefore>
<!ELEMENT currently_in_use      EMPTY>
<!ELEMENT dynamic_condition     valid?>
<!ATTLIST dynamic_condition
                        type PCDATA #REQUIRED
                        data CDATA #REQUIRED>

<!ELEMENT crl_test      yes_no_answer | dynamic_condition>
<!ELEMENT reval_test    yes_no_answer | dynamic_condition>
<!ELEMENT one_time_test now_answer | dynamic_condition>
<!ELEMENT renew_test    new_cert_answer |
                        dynamic_condition>
<!ELEMENT limit_test    dynamic_condition>
<!ELEMENT limit_status  (#PCDATA)>
<!ELEMENT service_status        (#PCDATA)>

<!ELEMENT test_definition ( crl_test | reval_test | one_time_test |
      renew_test | limit_test)>
<!ELEMENT test_definition_reply   reason>

<!ELEMENT status_query verbose?, valid?>
<!ELEMENT status_reply  (yes_no_answer, currently_in_use?) |
      now_answer |
(new_cert_answer, currently_in_use?) | limit_status, dy-
      namic_condition?>

<!ELEMENT delete_request valid>
<!ELEMENT delete_reply  reason>

<!ELEMENT signature     EMPTY>
<!ATTLIST signature     data CDATA #REQUIRED>

<!ELEMENT server_update cert, chain?, online_test_hash, de-
      lete_request*, test_definition*, status_query*, signature>
<!ELEMENT server_reply cert_hash, online_test_hash, de-
      lete_reply*, test_definition_reply*,
      status_reply*,service_status, signature
```

SPKI Performance and Certificate Chain Reduction

Yki Kortesniemi

Helsinki Institute for Information Technology
Helsinki University of Technology
P.O. Box 9800
FIN-02015 HUT
Yki.Kortesniemi@hiit.fi

Abstract: Authorisation certificate based access control owes much of its expressive power to delegation; delegation enables distributed access control management, where the authorisation decisions are manifested as certificate chains. Unfortunately, these chains have to be evaluated every time a right is used, and if the right is used repeatedly, this can result in significant performance overhead. However, if the chains are replaced with reduction certificates, this overhead can be cut down.

In this paper we discuss performance in SPKI and how it can be improved with certificate chain reduction. We elaborate on certificate chains, reduction certificates, and their performance implications, the choice of issuers of reduction, and take a look at the problems of reducing chains with online validity checks.

1 Introduction

Implementing a global service for a multitude of users can present daunting management challenges for the access control technology used. One solution is to use a technology that allows the management rights to be distributed along with the access rights as authorization certificates do. The Internet Engineering Task Force (IETF) has been developing Simple Public Key Infrastructure (SPKI) as a more flexible alternative to X.509 [El99a][El99b]. The key idea in SPKI is that anyone (or anything) with access to a resource can authorize others to use the resource by issuing them an authorisation certificate. Further, the authorisation certificates can be used to delegate the rights to other users without any help from the owner of the resource: users can delegate their own rights. These certificates therefore form chains, which always start from the verifier controlling access to the resource, go through 0-N intermediate entities (e.g. administrators) and end with the actual user of the resource. This means that it becomes possible e.g. for a global credit card service to authorize all of its regional offices to issue the actual credit "cards" to the end users and for the credit card users to create new credit cards that make it possible for children to use their parents credit right in such a way that the parents keep their own card and the children have a limit to the amount they can charge from the card [HL99].

A result of this process is that the user (e.g. the child) could end up with a long chain of certificates that has to be presented whenever the right is used – and storing, handling and evaluating long chains can result in significant performance overhead. In this paper we look at how this overhead could be reduced by using chain reduction certificates (or certificate result certificates), CRCs, that replace a chain of certificates with a single certificate having the same properties as the chain. The rest of the paper is organized as follows: Section 2 elaborates on the motivations for chain reduction; section 3 discusses reducing chains having online validations; section 4 talks about different reducers and section 5 presents our conclusions.

2 Motivations for Reducing Certificate Chains

The SPKI theory introduces the concept of a CRC – it is a certificate that corresponds to the semantics of the underlying certificates and online test results [El99a]. The main motivation for creating CRCs is performance benefits:

1. discovering the correct chain from a pool of certificates is not a trivial operation [Au98],

2. as neither the user's terminal nor the verifier always have storage space for long chains, some of the certificates might even have to be fetched from the network further adding overhead [HK00],

3. and even with the correct certificates, deducing the access decision from the rights expressed in the certificates present challenges [BD02].

By using a CRC, we can avoid repeating these costly operations and the verifier can instead evaluate a single certificate to reach the access decision. But here we also note that for a CRC to make sense from performance perspective, it normally has to be used repeatedly. More precisely, the cost of creating and using the CRC over its lifetime has to be less than the cost of using the chain (without creating the CRC) for the CRC to be beneficial. Naturally, it is not always possible to know in advance, whether a particular CRC will be used again in the future, but there has to be at least the possibility for that CRC to make sense. Another justification for a CRC can be, if the reduction (of a section of the chain) is created by someone other than the verifier thus freeing resources from the (potentially burdened) verifier. A third motivation for creating CRCs is to promote anonymity by hiding parties in the chain as proposed in [NKP99].

3 Reduction Certificates and Online Validations

SPKI structure draft [El99b] defines several online validity conditions used to limit the usage of the certificate. [KHS00] further adds a couple, one of which, limit, creates particular complications for reductions, as we shall soon see (limit is used to create certificate with a controlled amount of usage such as a credit card with a monthly quota or a bus ticket for 10 journeys – without limit these kind of applications are not possible).

In creating CRCs, there are two options: all the online validations can be performed before reduction, in which case the resulting certificate has no online conditions, but presumably a shorter validity period. The other option is to include some or all the online conditions in the CRC and let the verifier perform them as needed. However, there are problems in both approaches. It is not possible to perform all online validations in advance of usage. CRL and Reval can be performed in advance - their result is a validity period, which can be used to determine the validity period of the CRC. One-time and limit, on the other hand, have to be evaluated at the time of usage and therefore they have to be included in the CRC. Finally, due to the design of limit, it is not possible to perform a reduction over a certificate containing a limit condition, because that particular certificate has to be in the chain for the limit check to work.

A structural definition is required to include online tests from other certificates. The current SPKI structure does not define how CRCs are to be constructed, so the inclusion of online test

from the other certificates is still undefined. Nevertheless, the size of the CRC with online checks will be rather large, as we have to include complete certificates. Because the instructions to the online servers can be included in the s-part of the original certificate, we have to include the whole certificate to convince the online server that the instructions really come from the issuer. Just including the relevant validity part will not suffice, as there is no signature authenticating the information. With these limitations in mind, the performance improvements achievable with CRCs containing validity conditions are still an open question. Further performance improvements could be achieved, if all the remaining online validations in a CRC could be replaced with a single online validation representing all of them. Naturally, this raises trust issues, but could provide significant improvements, particularly in situation with limited network access. However, the other type of a CRC should still be very useful in many situations. Particularly, if no online validations are left, the resulting CRC can be quite fast to evaluate.

4 Different Reducers

The SPKI structure draft only talks about the verifier creating CRCs, possibly also for the benefit of others, who choose to trust this verifier. However, any other certificate issuer in the chain can also issue partial reductions starting from themselves and ending at any point after them in the chain. The largest reduction naturally comes from the original issuer until the final user. The trust issues in all the cases, where the reduction issuer is already a member of the chain, are fairly clear. As they simply use their existing right to issue certificates, no new parties are introduced, which could change the trust model. However, the reduction issuer has to be additionally trusted to make correct reductions. If the reduction carries fewer rights than the original chain, the original chain can still be used to get to the remaining rights, but this might be inconvenient or even impossible thus mandating a new reduction. If, on the other hand, the reduction carries more rights (larger amount or other/larger rights), the reducer is doing this at own expense – the original chain issuers can not be expected to take responsibility for this. Therefore, it always makes sense for the reduction issuer to keep a copy of the reduced chain so that any disputes can later be solved.

The cost of performing reductions are different for verifier and other issuers. The verifier would anyway have to check the chain and reduce it, so the additional effort of creating the CRC is not very large. The other issuers, however, would not normally evaluate the chain, so for them the additional effort is bigger. Therefore, not all issuers are like to offer reduction services for arbitrary users. The verifier, on the other hand, should probably always issue a CRC just in case (with the exception of chain without any remaining rights, naturally). The structure draft talks about other entities trusting the verifier for creating CRCs. This apparently implies that the verifier can act as a TTP creating reductions for others. If we accept TTPs, they would not necessarily even have to be verifiers; any suitable TTP could be used. But this changes the trust model of the system by introducing an outsider capable of creating certificates for anyone without limits (or at least the limits have to be much higher than for regular certificate issuers). Of course, a incorrectly acting TTP can be asked to justify the actions afterwards by presenting the original chains, but the TTP still creates a tempting target for attacks due to larger than normal rights.

5 Conclusions

We have discussed the role of certificate reduction certificates and the motivations for using them. Certificate chains are a product of the management process and should be viewed as such. We conclude that CRCs could provide performance improvements at minimal cost, if issued by the verifier. Finally, online validations still present challenges for reduction and should be further looked into.

Bibliography

[Au98] Aura, T.: Fast access control decisions from delegation certificate databases, in proceedings of 3rd Australasian Conference on Information Security and Privacy ACISP '98, July 1998, Brisbane, Australia

[BD02] Bandmann, O.; Dam, M.: A Note on SPKI's Authorisation Syntax, Proceedings of 1st Annual PKI Research Workshop, April 2002, Maryland, USA

[El99a] Ellison, C.; Franz, B.; Lampson, B.; Rivest, R.; Thomas, B.; Ylönen, T.: SPKI certificate theory. Request for Comments: 2693, September 1999.

[El99b] Ellison, C.; Franz, B.; Lampson, B.; Rivest, R.; Thomas, B.; Ylönen, T.: Simple public key certificate. Internet draft (expired), IETF SPKI Working Group, July 1999.

[HK00] Hasu, T.; Kortesniemi, Y.: Implementing an SPKI Certificate Repository within the DNS, Poster Paper Collection of the Theory and Practice in Public Key Cryptography (PKC 2000), January 2000, Melbourne, Australia

[HL99] Heikkilä, J, Laukka, M: SPKI based Solution to Anonymous Payment and Transaction Authorization, Proceedings of the 4th Nordic Workshop on Secure IT Systems, 1999, Kista, Sweden

[KHS00] Kortesniemi, Y.; Hasu, T.; Särs, J.: A Revocation, Validation and Authentication Protocol for SPKI Based Delegation Systems, Proceedings of Network and Distributed System Security Symposium, February 2000, San Diego, California

[NKP99] Nikander, P.; Kortesniemi, Y.; Partanen, J.: Preserving Privacy with Certificates in Distributed Delegation, Proceedings of 1999 International workshop on Practice and Theory in Public Key Cryptography, March 1999, Kamakura, Japan