

## Validity Management in SPKI

Yki Kortensniemi  
Helsinki University of Technology  
Yki.Kortensniemi@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):

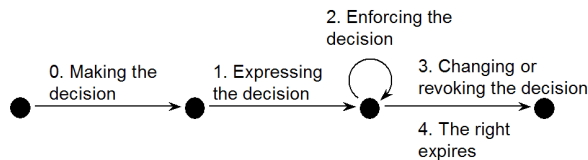


Figure 1. Phases of access control.

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.

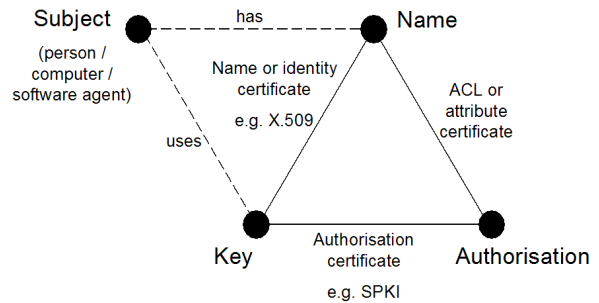


Figure 2. Three major types of certificates.

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 situations, 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 systems, 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 are 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 re-

voke 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).

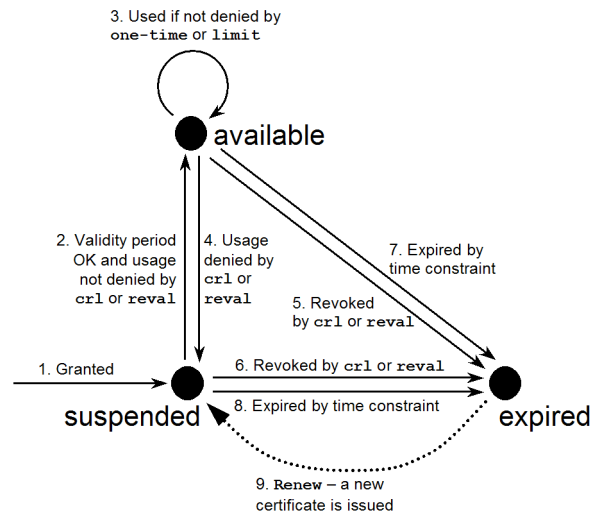


Figure 3. The lifecycle of an SPKI certificate.

## 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 re-issue 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. Zucherato: 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 Identifiers (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 Kortnesniemi, 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 Kortnesniemi, 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 Proceedings 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.

```
<!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-
                        delete_request*, test_definition*, status_query*, signature>
<!ELEMENT server_reply  cert_hash, online_test_hash, de-
                        delete_reply*, test_definition_reply*,
                        status_reply*, service_status, signature
```

## 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>
```