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>?  "("
        "canceled" <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> ")"
                "invalid"?  <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>?
                "renew" <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> ")"
                "invalid"?  "(" "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 5. 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

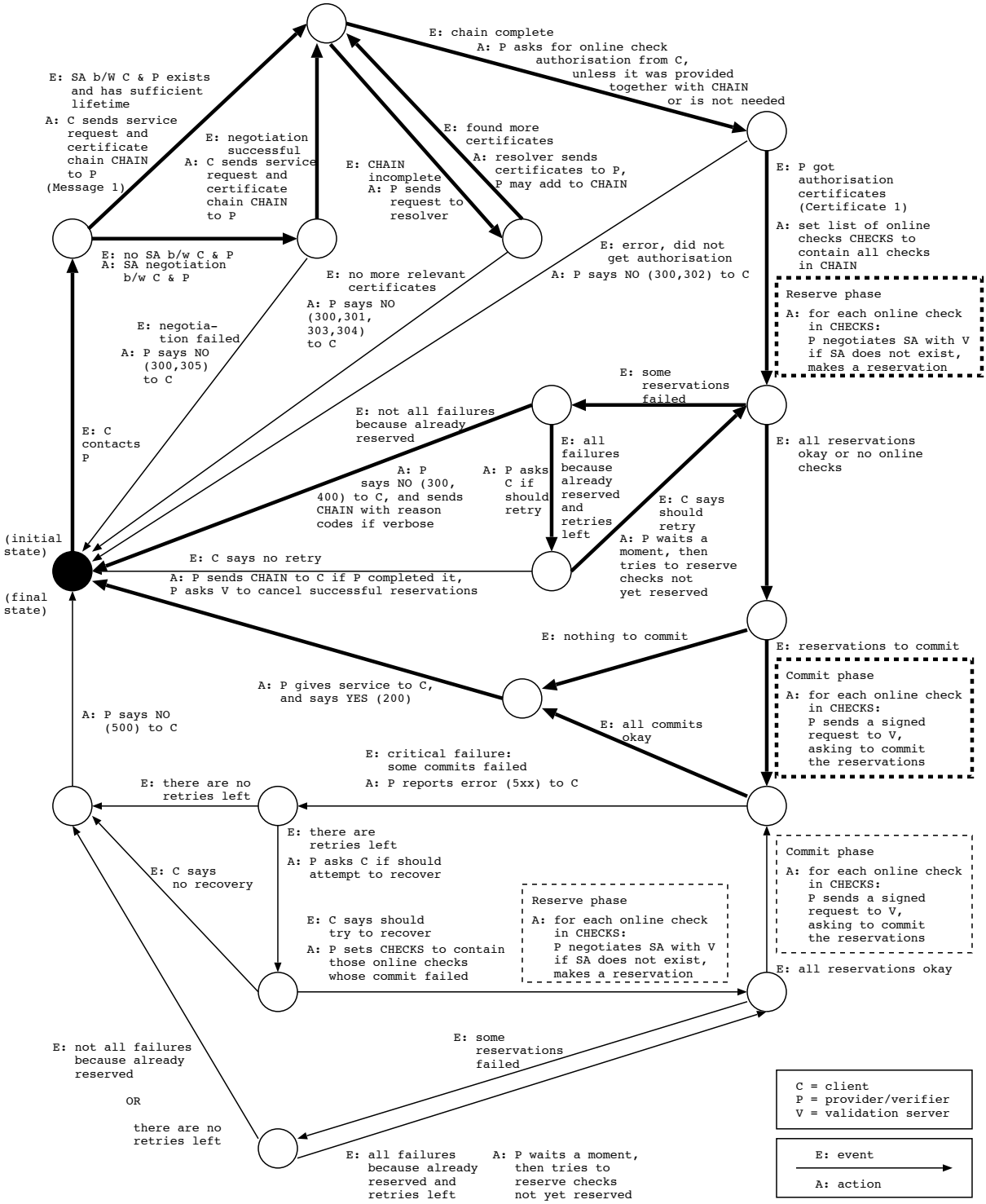

Figure 1: Verifier state machine.

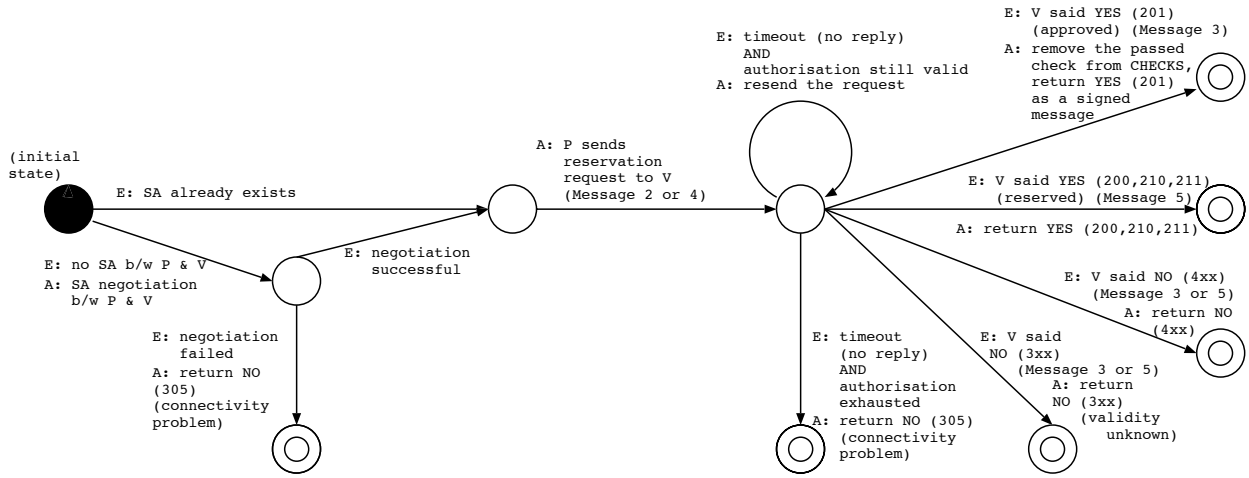

Figure 2: The reserve phase. Executed concurrently for each online check in the list of checks $CHECKS$.

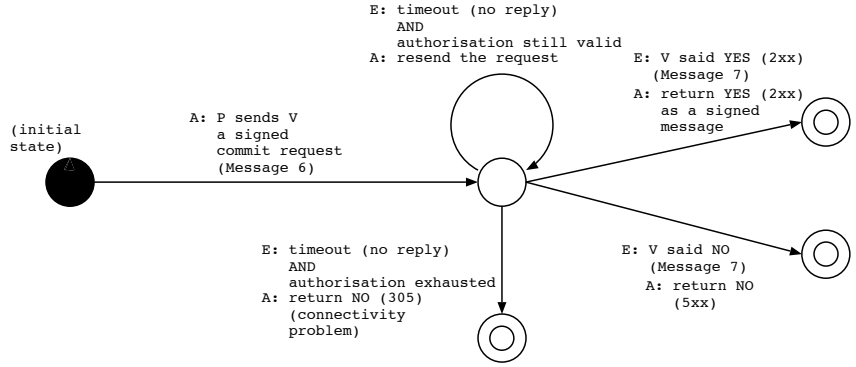

Figure 3: The commit phase. Executed concurrently for each online check in the list of checks $CHECKS$.

or fail. In the first message the verifier announces the wish to use some of the limit:

(Message definition 4)

```
(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.

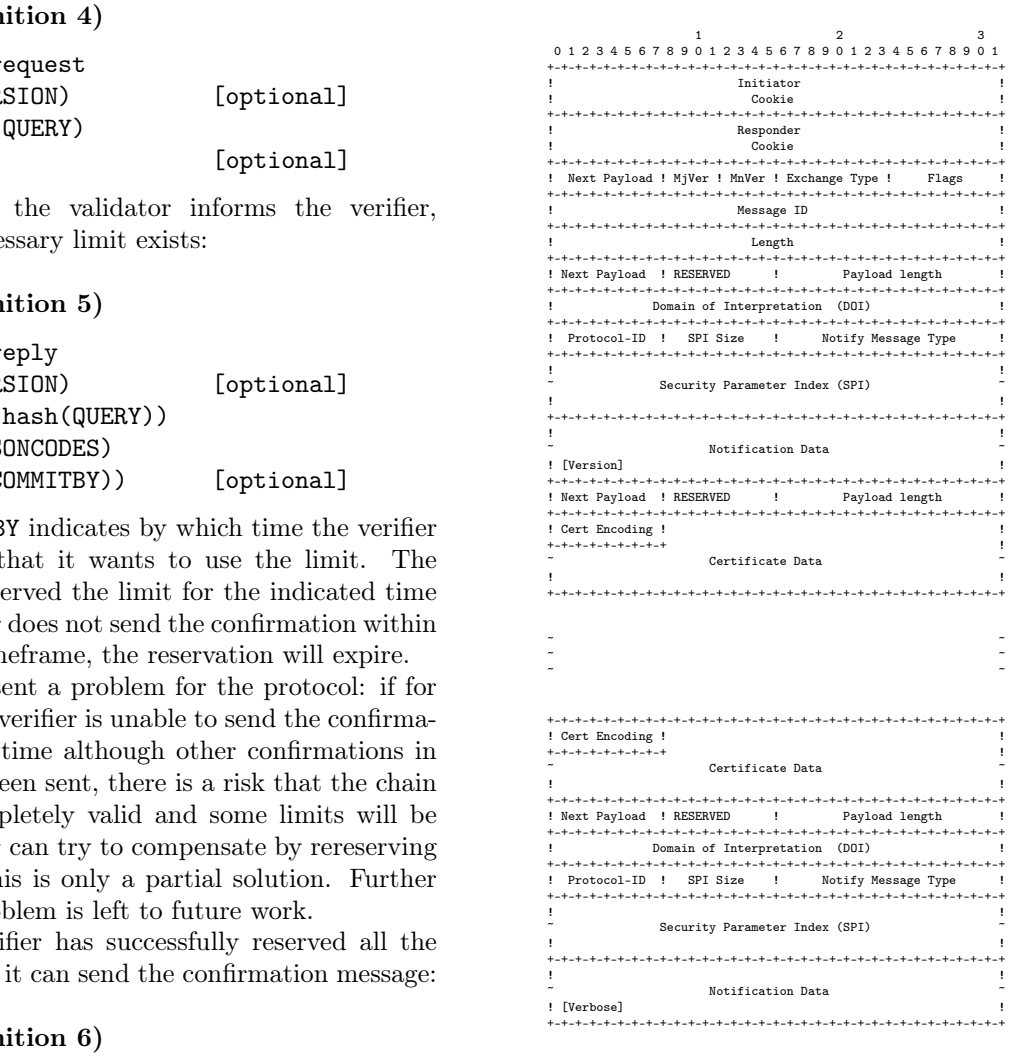

Figure 4: An ISAKMP payload definition of a service request.

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.

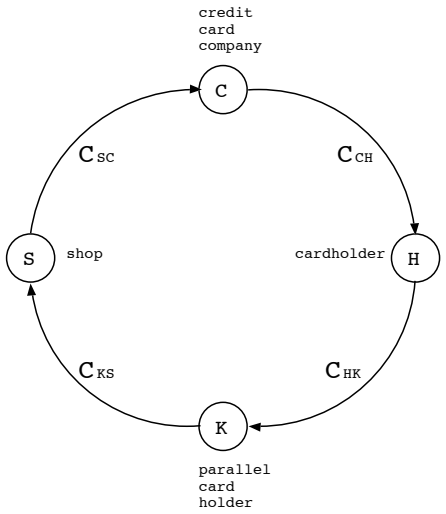

Figure 5: An example scenario.

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.)

$$c_{CH} = \texttt{(cert (issuer } C\texttt{) (subject } H\texttt{) (propagate) (tag (has-credit unlimited)) (not-after } E\texttt{))} \quad \textbf{(2)}$$

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 stor-

age 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 YlÃűnen. 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 YlÃűnen. 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.