This draft suggests the use of XML-encoding for SPKI certificates. The main purpose of this draft is to define the XML-encoding, not to present new ideas about SPKI. The SPKI theory is explained in [RFC2693] and the s-expression structure that has been the basis for this draft, is defined in [SPKI]. As an exception to the above mentioned, this draft discusses a problem in tag processing rules presented in [SPKI] and suggests an improvement.

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Distribution of this document is unlimited. Comments should be sent to the author (juha.paajarvi@firsthop.com).
Abstract

This draft suggests a standard form for encoding SPKI certificates in XML as opposed to the original s-expression encoding defined in [SPKI]. The standard form is defined as an XML document type definition (DTD). The main emphasis is on the XML-encoding of an authorization certificate that is the basic SPKI certificate form.

This draft provides also a brief introduction to XML and a short discussion about the benefits of choosing XML as the certificate encoding format.

In addition, this draft discusses the problems of automatic processing of tags (attributes and authorizations transferred by a certificate are called tags in SPKI) when reducing certificates. An example of encoding Java permissions in an SPKI certificate is given to demonstrate the problem and, finally, a solution to this problem is suggested.
Table of Contents

1 Introduction ................................................... 4
1.1 Scope of this Document .................................... 5
1.2 Glossary of Terms ........................................ 5
2 XML Basics .................................................... 8
2.1 Well-Formed XML Documents ................................ 8
2.2 Valid XML Documents ...................................... 8
2.3 Canonical Form ............................................ 10
2.4 Encoding Binary Data ..................................... 10
3 Primitive Objects ............................................ 11
3.1 Public-key .................................................. 11
3.1.1 RSA Public-key ........................................ 11
3.1.2 DSA Public-key ........................................ 11
3.2 Hash ........................................................ 11
3.3 Hash of Public-key ........................................ 12
3.4 Uniform Resource Locator (URI) ............................ 12
3.5 Signature ................................................... 13
4 Authorization Certificate ..................................... 14
4.1 Version .................................................... 14
4.2 Issuer ..................................................... 15
4.3 Subject ..................................................... 15
4.3.1 Object-Hash ............................................ 16
4.3.2 Keyholder .............................................. 16
4.3.3 Threshold ............................................... 16
4.4 Delegation ................................................ 17
4.5 Tag ........................................................ 17
4.6 Validity ................................................... 18
4.6.1 Date Format ............................................ 18
4.6.2 Online Test ............................................ 18
4.6.3 New-cert ............................................... 19
4.7 Comment ................................................... 20
5 Name Certificate .............................................. 21
6 Other SPKI Objects ............................................ 22
6.1 ACL ........................................................ 22
6.2 CRL and Delta-CRL ....................................... 22
6.3 Revalidation ............................................... 22
6.4 One-Time .................................................. 22
7 Tag Intersection and Java Permissions ...................... 23

Appendix A - The full DTD ...................................... 25
Appendix B - Examples of XML-encoded Certificates .......... 27

References ....................................................... 29
Acknowledgements ............................................... 30
Author’s Address ................................................ 30
1 Introduction

Certificates are needed as a part of security architecture in many applications. Because of their complexity, only few developers are able to implement certificate tools and utilize certificates in their own applications. Certificate tools are also very expensive due to the complexity of certificate technology.

Cryptography is surely the most difficult technology that must be mastered in implementing certificate tools. Another difficulty often faced by programmers is the encoding format required for certificates. X.509 certificates are encoded as ASN.1 data structures, and SPKI certificates utilize s-expressions as their encoding format. Neither of these encoding formats are widely used in the Internet. This raises the question: "Could there be a simpler, more widely used format for encoding certificates?".

This draft suggests the use of XML as the encoding format for certificates and represents a standard form for encoding SPKI authorization certificates in XML.

There are a number of reasons why XML is a good choice for encoding certificates:

- Despite its relatively young age, XML has been quickly adopted. Free, high quality XML software packages are available. These packages can encode, parse and verify XML documents.

- Even very small devices like WAP-capable mobile phones and 3rd generation mobile phones will most likely have XML parser built into the platform.

- There are two APIs that can be used to interface an XML processor: Document Object Model [DOM] and Simple API for XML (SAX). As a consequence, developers are free to change the XML processor of a software with relatively low cost as long as both the new and the old XML processors provide the same API.

- XML certificate documents can be verified against the DTD definition by any verifying XML parser. The possibility for automatic verification makes it easy to verify that an implementation produces certificates that are structured as required by this specification.

- XML is both human and machine readable format. This makes certificates more explicit to the users and developers of computer systems.

- XML documents can easily be converted to different forms of representation with XSL.
The real question seems to be: "Why should any other encoding than XML be used for certificates?". There will, of course, be objections to the size of XML-encoded certificates compared to a binary-encoded certificates. However, reality is that memory capacity is not a problem now and will not be in the future. Lots of bandwidth will be available for even the smallest gadgets because they are supposed to handle at least sound, if not video.

The processing requirements of XML documents are slightly higher than those of binary-encoded ones. However, this will not be a problem, because certificate processing includes, for example, the validation of a digital signature that is a slower operation by an order of magnitude. And, in the end, a normal device will not handle more than a couple of certificates in a minute.

1.1 Scope of this document

This specification describes the structure and the format for encoding SPKI certificates in XML. The actual semantics and theory of SPKI certificates are discussed in more detail in RFC 2693 "SPKI Certificate Theory" [RFC2693]. The already expired draft "Simple Public Key Certificate" [SPKI] defines s-expression structures for SPKI certificates.

Basically, this draft discusses SPKI as it is presented in [SPKI]. The major difference is the use of XML-encoding. In addition, a problem in SPKI tag processing is discussed and a solution is proposed.

The main content of this draft is the XML-encoding of SPKI authorization certificates. Authorization certificates are the most important SPKI certificate structure. XML DTD definitions are also given for name certificates, ACLs, CRLs and online test reply formats, but these structures are not explained in this draft. [SPKI] explains the semantics of these objects. This version does not have any definition of sequence formats presented in [SPKI]. These shortages may be fixed in the next versions of this draft.

Unlike [SPKI], this document does not describe 5-tuple reduction rules. The descriptions in [RFC2693] and [SPKI] are considered to be sufficient.

1.2 Glossary of Terms

In this section the terms used in this document are defined. Most of these definitions are taken directly from [RFC2693] and [SPKI].

ACL: an Access Control List: a list of entries that anchors a
certificate chain. Sometimes called a "list of root keys", the ACL is the source of empowerment for certificates. That is, a certificate communicates power from its issuer to its subject, but the ACL is the source of that power (since it theoretically has the owner of the resource it controls as its implicit issuer). An ACL entry has potentially the same content as a certificate body, but has no Issuer (and is not signed). There is most likely one ACL for each resource owner, if not for each controlled resource. [RFC2693]

CERTIFICATE: a signed instrument that empowers the Subject. It contains at least an Issuer and a Subject. It can contain validity conditions, authorization and delegation information. Certificates come in three categories: ID (mapping <name,key>), Attribute (mapping <authorization,name>), and Authorization (mapping <authorization,key>). An SPKI authorization or attribute certificate can pass along all the empowerment it has received from the Issuer or it can pass along only a portion of that empowerment. [RFC2693]

FULLY QUALIFIED NAME: a local name together with a global identifier defining the name space in which that local name is defined. [SPKI]

GLOBAL IDENTIFIER: a globally unique byte string, associated with the keyholder. In SPKI this is the public key itself, a collision-free hash of the public key or a Fully Qualified Name. [SPKI]

HASH: a cryptographically strong hash function, assumed to be collision resistant. In general, the hash of an object can be used wherever the object can appear. The hash serves as a name for the object from which it was computed. [SPKI]

ISSUER: The signer of a certificate and the source of empowerment that the certificate is communicating to the Subject. [RFC2693]

KEYHOLDER: the person or other entity that owns and controls a given private key. This entity is said to be the keyholder of the keypair or just the public key, but control of the private key is assumed in all cases. [RFC2693]

NAME: a SDSI (and an SPKI) name is always relative to the definer of some name space. This is sometimes also referred to as a local name. A global (fully qualified) name includes the global identifier of the definer of the name space. [SPKI]

ONLINE TEST: one of three forms of validity test: (1) CRL; (2) revalidation; or (3) one-time revalidation. Each refines the date range during which a given certificate or ACL entry is considered valid, although the last defines a validity interval of effectively zero length. [SPKI]

PRINCIPAL: A cryptographic key, capable of generating a digital
signature. This document deals with public-key signatures but any
digital signature method should apply. [RFC2693]

PROVER: the entity that wishes access or that digitally signs a
document. The Prover typically sends a message or opens a channel to
the Verifier that then checks signatures and credentials sent by the
Prover. [SPKI]

SPEAKING: A Principal is said to "speak" by means of a digital
signature. The statement made is the signed object (often a
certificate). The Principal is said to "speak for" the Keyholder.
[RFC2693]

SUBJECT: The thing empowered by a certificate or ACL entry. This can
be in the form of a key, a name (with the understanding that the name
is mapped by certificate to some key or other object), a hash of some
object, or a set of keys arranged in a threshold function. [RFC2693]

THRESHOLD SUBJECT: A Subject for an ACL entry or certificate that
specifies K of N other Subjects. Conceptually, the power being
transmitted to the Subject by the ACL entry or certificate is
transmitted in (1/K) amount to each listed subordinate Subject. K of
those subordinate Subjects must agree (by delegating their shares
along to the same object or key) for that power to be passed along.
This mechanism introduces fault tolerance and is especially useful in
an ACL entry, providing fault tolerance for "root keys". [RFC2693]

XML: Extensible Markup Language defined by W3C; a meta language that
can be used to describe structured documents. XML can be used to
define languages that in turn can be used to write structured
documents. XML is a subset of SGML, the Standard Generalized Markup
Language. The current version of XML is defined in [XML].

DTD: Document Type Definition is an XML document which defines a
grammar for a class of documents. Most importantly, a DTD declares
the elements (sometimes called tags) that encode the structure of a
document. A validating XML software can verify the validity of a
document by comparing the document to its DTD. See [XML] for more
precise definitions.

XSL: Extensible Stylesheet Language is a language closely related to
XML. It defines the means to convert an XML document to other
representation forms. For example, XSL rules which convert an
XML-encoded authorization certificate to HTML or LaTeX
representation, could be defined.

BASE64: An encoding method to convert binary data to and from an
ASCII representation. Base64 transformation is defined in [RFC2045].
2 XML Basics

All the definitions in this specification are presented using the standard XML document type definitions (DTD). The DTD is based on XML 1.0 Specification [XML], which also gives a detailed technical specification of XML. In addition, there are a number of good books about XML (and DTD). Here, only the XML basics are explained.

XML documents consist of elements that encode the structure of an XML document. An element starts with a start-tag, which is marked by putting the element name inside angle brackets. Element contains all the data until the end-tag for the element is reached. End-tag is similar to start-tag, except that the element name inside the angle brackets is preceded by a slash (/). Elements can contain other elements or plain text data. Here is a simple example of an XML document:

```xml
<example>
  <text>Example of an XML document</text>
</example>
```

Here, the start-tag of the example element is on the first line. The example element contains another element called "text". The text element contains in turn textual data, or, in XML terms "parsed character data". Both example and text elements are closed with end-tags.

2.1 Well-Formed XML Documents

The above XML document is well-formed. This is important because an XML document must be well-formed if it is to be processed. For an XML document to be well-formed, it must fulfill the following requirements:

1. It has one or more elements.
2. It has a root element that does not belong to the content of any other element.
3. For other elements, if the start-tag is inside another element, the end-tag must be inside the same element.

See section 2.1 of [XML] for an exact definition of well-formedness.

2.2 Valid XML Documents

Validity is a bigger requirement for an XML document than well-formedness. For an XML document to be valid it MUST be well-formed and MUST conform to some XML DTD (Document Type Definition). The key here is DTD. A DTD defines the structure that
some class of documents must follow. Let’s take a brief look at DTDs with the above example document. The following DTD defines the structure that must be followed when writing examples of the same type. The DTD looks like this:

```xml
<!ELEMENT example (text)>
<!ELEMENT text (#PCDATA)>
```

This definition states that an example element must contain a text element. It also states that the text element must contain parsed character data (PCDATA) that is, plain ASCII text. In a DTD, an element declaration always starts with "<!ELEMENT " followed by the name of the element being declared. The element name is followed by a comma separated list of elements that must be contained inside the declared element in a valid XML document. The order of elements in the list is significant.

An element can be declared optional by adding "?" after its name. This means that there must be 0 or 1 occurrences of that element. "*" can be used to specify closure (0 or more occurrences) and "+" to specify non-empty closure (1 or more occurrences). "|" is a logical OR, that is, if "|" is a separator in a list, then exactly one of the elements of the list must occur in a document obeying the rule. Examples of these can be seen in appendix A which contains the DTD for XML encoded SPKI authorization certificates.

An XML document can also contain empty elements, i.e. elements that do not have any content. Usually an empty element signals something if it is present in a document. This specification defines the following empty element:

```xml
<delegation/>
```

As can be seen, an empty element combines start and end-tags to a single tag where a slash (/) is put after the element name. This is defined in the DTD as:

```xml
<!ELEMENT delegation EMPTY>
```

In addition, a DTD can contain attribute declarations for elements. Attributes are name-value pairs defining properties of an element. The text element defined above could have an attribute defining the language of the text enclosed in the element. This could be defined as follows:

```xml
<!ATTLIST text language CDATA #IMPLIED>
```

This means that the text element has an attribute called language and the type of the attribute value is character data. The text "#IMPLIED" in the declaration states that the language attribute is
not required in a valid document. Changing "#IMPLIED" to "#REQUIRED"
would make the language attribute mandatory. By using the language
attribute, the example presented above gets the following form:

```xml
<example>
  <text language="English">
    Example of an XML document
  </text>
</example>
```

### 2.3 Canonical Form

XML documents can be modified extensively without changing the
meaning of the document. One can, for example, modify indentation of
an XML document or add or remove line breaks. Still, the document is
considered equal to the original (by most applications). This causes
problems when calculating hashes (and digital signatures) of XML
documents.

In order to make the calculation of hashes possible, a canonical form
of XML must be specified. Canonical form eliminates all
possibilities for optional representations of an XML document (or an
XML element and its contents). In this draft, canonicalization is
needed mainly to calculate a digital signature for a certificate.
Since the signature will use the format that is being defined by the
W3C/IETF joint XML-Signature working group, any canonicalization
method defined in [XMLdsig] can be used. At the moment [XMLdsig]
specifies a minimal canonicalization and a complete (and complex)
one [XML-c14n].

### 2.4 Encoding Binary Data

XML documents can contain only textual (ASCII) data and, thus, all
binary data within a certificate must be encoded using base64
encoding. For example, public-keys and hashes are objects that
contain binary data and must thus be encoded. The base64 conversion
must be done as specified in [RFC2045].

3 Primitive Objects

XML encoded SPKI certificates build on a number of primitive objects. This chapter defines and explains these primitives.

3.1 Public-key

```
<!ELEMENT public-key (rsa-pubkey|dsa-pubkey)>
```

Public-key element specifies the public-key of a subject or an issuer. At this stage, two public-key types are specified: RSA and DSA. New key types can be specified later, if necessary.

Note that contrary to [SPKI], a public-key element does not include URIs pointing to certificates empowering the public-key. These URIs are part of the element containing the public-key element (i.e., issuer and subject elements).

3.1.1 RSA Public-key

```
<!ELEMENT rsa-pubkey (rsa-e,rsa-n)>
<!ELEMENT rsa-e (#PCDATA)>
<!ELEMENT rsa-n (#PCDATA)>
```

RSA public-key consists of the public exponent E and the modulus N. Parameters E and N must be base64 encoded and enclosed in rsa-e and rsa-n elements, respectively.

3.1.2 DSA Public-key

```
<!ELEMENT dsa-pubkey (dsa-p,dsa-q,dsa-g,dsa-y)>
<!ELEMENT dsa-p (#PCDATA)>
<!ELEMENT dsa-q (#PCDATA)>
<!ELEMENT dsa-g (#PCDATA)>
<!ELEMENT dsa-y (#PCDATA)>
```

DSA public-key consists of prime P, subprime Q, generator G and the actual public-key value Y. Parameters P, Q, G and Y must be base64 encoded and enclosed in dsa-p, dsa-q, dsa-g and dsa-y elements, respectively.

3.2 Hash

```
<!ELEMENT hash (#PCDATA)>
<!ATTLIST hash hash-alg CDATA #REQUIRED>
```
Hash element defines a hash value. The hash algorithm that has been used to calculate the hash MUST be specified with the hash-alg attribute. The predefined values are "md5" [RFC1321] and "sha1" [SHA-1]. Other algorithms can be used as long as the algorithm type is specified with an URL pointing to the description of the algorithm.

3.3 Hash of Public-key

<!ELEMENT hash-of-key (hash,uri*)>

Hash-of-key element refers to a public-key of an issuer or a subject by the hash of that key. The main purpose of this element is to shorten certificates. It can also be used to protect a weak public-key against cryptanalysis. (An entirely different issue is the purpose of a public-key, if it is too weak to be made public.)

In case the public-key of an issuer or a subject is specified with the hash of the actual key, the identification of the key itself must be easy. An URI (or a set of URIs) can be specified inside the hash-of-key element to identify or locate the key being referred. The hash of the key can then be used to verify that the key is the one being referred. The key can also be identified directly with its hash if, in an application context, it can be found based on its hash.

When calculating a hash of a public-key, the public key must first be encoded as a public-key element (see section 3.1) in a canonical format and then hashed. The steps to calculate the hash of a public-key are the following:

1. Encode the public-key element so that there are no line breaks and no superfluous white spaces.
2. Apply the minimal canonicalization defined in [XMLdsig].
3. Calculate the hash over the canonicalized public-key element.

The step 1 is needed to make the use of the minimal canonicalization possible in the calculation of a hash of a public-key.

3.4 Uniform Resource Locator (URI)

<!ELEMENT uri (#PCDATA)>

URI is an uniform resource identifier as specified by [RFC2396]. URI specifies a generic standard form for identifying any resource. The most widely used form of URI is URL (Uniform Resource Locator) that identifies the resource via specifying its network location.
In a certificate, URIs usually refer to related objects such as other certificates. The meaning of the object referred by URI is always specified by the context. For example, an URI in an issuer element can point to certificates empowering the issuer.

### 3.5 Signature

A digital signature is needed for most objects presented in this draft. These include authorization certificate, name certificate, CRL, delta-CRL, revalidation reply and one-time online test reply. The signature for these objects MUST be the XML signature as specified by [XMLdsig].
Authorization Certificate

Authorization certificate is the basic SPKI certificate form. It transfers a set of authorizations, permissions or attributes (also called tags) from a certificate issuer to the subject of a certificate.

This chapter defines an XML encoding for SPKI authorization certificates and explains the semantics of each certificate element. The complete DTD can be found in appendix A.

SPKI authorization certificates are used mainly to make authorization decisions. These decisions are done by verifying the certificate of the entity whose authorization is being resolved. In fact, to check authorizations, usually a whole chain of certificates must be verified. However, because authorization certificates just transfer authorizations, verifying a chain does not yet authorize anyone; the authorizations must originate somewhere. In SPKI, authorizations must originate from the verifier itself and they must be transferred through a certificate chain to the subject of the last certificate. This is usually described by saying that the authorization forms a circuit: it originates from the verifier and flows through a certificate chain to a subject, who then presents the certificates to the verifier in order to get access rights, for example.

In SPKI, the structure containing the initial authorizations (that can be transferred by certificates) is called ACL. ACL is an internal data structure kept in the trusted memory of the verifier and it is never communicated to anyone. Thus, the ACL structure does not have to be standardized, but SPKI defines an ACL to be a list of authorization certificates that do not have an issuer or a signature. Example XML encoding for an ACL is defined in section 6.1.

The DTD definition for XML encoding of an SPKI authorization certificate is the following:

```xml
<!ELEMENT cert (version?, issuer, subject, delegation?, tag, validity, comment?)>
```

This defines the cert element that contains all the necessary information of an authorization certificate (although, it does not contain a signature; it is a separate element). The most important elements of an authorization certificate are the elements usually referred to as the 5-tuple: issuer, subject, delegation, tag and validity. All these elements are discussed in more detail in the following sections.

4.1 Version
<!ELEMENT version (#PCDATA)>

Version number must be defined as an alphanumeric string. The version number for the certificate structure defined in this draft is 0. If the version element is missing, the certificate version is 0. If the version string is not recognized, the certificate must be ignored.

4.2 Issuer

<!ELEMENT issuer ((public-key|hash-of-key),uri*)>

Issuer element defines the entity that has issued the certificate. Issuer is always identified either by a public-key or a hash of a public-key. SPKI uses also the term ‘principal’ of public-keys and hashes of keys.

The optional URIs can refer to certificate(s) that authorize the issuer to transfer the authorizations defined in the certificate. Usually, the prover is expected to present all the necessary certificates to the verifier. However, this location information is allowed for flexibility. Anyone trying to track down certificates forming an authorization chain can use these URIs to locate the certificates.

4.3 Subject

<!ELEMENT subject ((public-key|hash-of-key|name|object-hash|keyholder|threshold),uri*)>

Subject element defines the subject of the certificate. Subject can be described as the owner of the certificate; it is the entity empowered by the certificate. The subject is identified by any of the following subject types: a public-key, hash of a public-key, name, hash of an object, keyholder or K-of-N threshold of subjects.

In the most basic form, if the other subject types are unnecessary for the application intent, the following definition of subject element can be used:

<!ELEMENT subject ((public-key|hash-of-key),uri*)>

URIs can be specified also for the subject element. These URIs can for example, provide the location of some additional information about the subject. If for example, the subject is identified with a name, an URI could refer to a name certificate for that name. Note that if the subject is identified by a hash-of-key, the URI that refers to the actual key must be inside the hash-of-key element.
The following sections explain all the different subject types, except public-key and hash-of-key, which were explained in chapter 3.

4.3.1 Object-Hash

<!ELEMENT object-hash (hash,uri*)>

Object hash is a subject type that refers to a data object other than a public-key or hash-of-key. The actual data object can be any application specific data that is being authorized or given attributes by the certificate. It could be, for example, an executable program for which the certificate transfers permissions.

To make the identification of the referred object easy, an URI (or a set of URIs) can be specified inside the object-hash element to identify or locate the object. The hash of the object can then be used to verify that the object is the one that is being referred to.

4.3.2 Keyholder

<!ELEMENT keyholder (public-key|hash-of-key|name)>

Keyholder subject refers to a flesh and blood subject, such as a human being. SPKI defines a certificate with this subject type to be a message to a human rather than a certificate that is a part of an automated verification process. In other words, the keyholder subject type is of very little interest to verifying code.

4.3.3 Threshold

<!ELEMENT threshold (subject+)>
<!ATTLIST threshold k CDATA #REQUIRED>

Threshold subject specifies n subjects, of which k must agree before permissions are passed along. N is the number of subjects inside the threshold element and k is defined explicitly by the mandatory attribute k. The value of the attribute k MUST be a number and k MUST be lower or equal to n. Note that [SPKI] requires n to be defined explicitly, but this is not the case in this specification. Defining n as an attribute would only make it possible to have certificates where n would not be equal to the number of subjects.

Threshold subject is a complex construct that allows, for example, to build ultra-secure applications where multiple subjects are required to agree before some critical operation can be executed. See [SPKI] for extensive description and discussion about threshold subject.
4.4 Delegation

<!ELEMENT delegation EMPTY>

If the body of a certificate contains a delegation element, the subject of the certificate can not only use the permissions or attributes transferred by the certificate, but also has the permission to transfer them (or part of them) further to others.

4.5 Tag

Tag element contains the attributes, authorizations, permissions, capabilities, parameters or whatever things a certificate transfers.

A DTD definition is not given for the tag element. This is because tags of any name can be used inside a tag element. There are, however, some structures that can be used inside a tag element. Possible tag structures are listed below.

If the certificate transfers all possible tags, the form of the tag element is following:

<tag>*</tag>

If this is not the case, the tag element must name the tag that is transferred. In the simplest form, the transferred tag is simply stated. For example:

<tag>socket allow dest .com</tag>

This could mean for some application that socket connections to .com domain internet hosts are allowed. A certificate with this tag transfers just the mentioned permission. If a set of tags is to be transferred, the following form SHOULD be used:

<tag>socket allow dest <set>.com .org .net</set></tag>

In this case, connections would be allowed to .com, .org and .net domains. Another possibility would be to specify a prefix. For example:

<tag>socket allow dest <prefix>www.</prefix></tag>

This would allow socket connections to internet hosts whose name starts with "www.". [SPKI] specifies also a range matching for tags. For example:

<tag>socket allow port <range ordering="numeric" lte="le" low="20" gte="ge" up="80" /></tag>
[SPKI] defines that the possible values for ordering attributes are "alpha", "numeric", "time", "binary" and "date". The lower and upper limits are optional, but if a lower limit is present, then also the type of lower limit comparison, the "lte" attribute, must be specified. The accepted values for lte are "l" (lower) and "le" (lower or equal). The same applies to the upper limit. Accepted values for gte are "g" and "ge".

When two certificates are reduced to one, the tags of these two certificates are intersected. Instructions for calculating the intersection are given in [SPKI]. In SPKI, tags are always intersected automatically by a generic certificate processor when certificates are reduced. The tag grammar has been designed to allow automatic processing of tags. The intention has been good, but the grammar introduced above causes problems when the sets and ranges of tags cannot be expressed in simple, application independent means. For example, this is the case for Java permissions. Chapter 7 suggests a solution to this problem.

4.6 Validity

<!ELEMENT validity (notbefore?, notafter?, online*, new-cert?)>

The validity element specifies validity constraints of a certificate. The basic constraints are "not before" and "not after" dates. These dates specify the maximum validity interval for the certificate. The certificate cannot be used before the "not before" date and after the "not after" date. If "not before" is not specified, the lower bound of the validity interval is considered to be negative infinity. This can be used if the certificate takes effect right away when it is granted. Similarly, if "not after" is not specified, the validity interval extends to infinity, i.e. the certificate is valid forever. Not specifying "not after" is strongly discouraged. As a rule of thumb, certificates should never be given unlimited lifetime.

4.6.1 Date Format

The "not before" and "not after" dates MUST be expressed in the format YYYY-MM-DD_HH:MM:SS. The times are always in UTC. The DTD definition for these elements are:

<!ELEMENT notbefore (#PCDATA)>
<!ELEMENT notafter (#PCDATA)>

4.6.2 Online Test

As mentioned before, the validity interval specified by "not before"
and "not after" dates is the maximum validity interval for a certificate. This interval can be further restricted with online tests. An online test can shorten the lifetime or even invalidate the certificate. An online test can be used to react to changed circumstances or to achieve fine grain control of the validity of the certificate. Online test is expressed with the format:

```xml
<!ELEMENT online (uri*, (public-key|hash-of-key), params)>
<!ATTLIST online type (crl|reval|one-time) #REQUIRED>
```

The URIs specify where an online test can be found. If there is more than one URI, any one of them can be used. The result of an online test is a response digitally signed by the public-key contained (or referred to) in the online test element. The params element can contain any parameters that MUST be passed to the online test.

The type attribute defines the online test type in question. Currently there are three different types of online tests: certificate revocation list (CRL), online revalidation and a one-time test. The response of an online test contains a validity interval that MUST be intersected with the validity interval of the certificate that contains the online tests.

A certificate revocation list is a kind of "black list" of certificates. Certificates listed in a CRL are considered to be invalid. More about the theory and subjects behind CRLs can be found in [RFC2693]. CRL format will be specified later.

Revalidation is an online test that gives a positive statement about validity of a certificate (unlike CRL which gives negative statement). The result of a revalidation online test is usually a stricter validity period for the certificate than was specified by "not before" and "not after" dates.

When one-time online test is carried out successfully, it states that the certificate is valid "now". Because of this, this online test must always be done by the verifier. See [SPKI] for more details about online tests.

4.6.3 New-cert

```xml
<!ELEMENT new-cert (uri*)>
```

The new-cert element can be used to specify a location (or multiple locations) from where a new copy of the certificate in question can be fetched. The new-cert element is meant to be used in certificates that have a short life-time. The new certificate MUST be fetched only after the current certificate (the one that contains the new-cert element) has expired.
4.7 Comment

<!ELEMENT comment (#PCDATA)>

The optional comment field can be used to attach short messages to a certificate. The content of a comment element is intended for a human rather than an automatic certificate processor.
5 Name Certificate

Name certificate encoding is similar to the one presented in [SPKI]. This draft version gives only the DTD definition of a name certificate. Readers should see [SPKI] for more details about name certificates and their reduction.

The DTD definition for name certificate is:

```
<!ELEMENT name-cert (version?,name-issuer,subject,valid?,comment?)>
<!ELEMENT name-issuer ((public-key|hash-of-key),name)>
```

The name element is defined as follows:

```
<!ELEMENT name ((public-key|hash-of-key)?,#PCDATA)>
```

If a public-key or a hash of a public-key (a principal) is not defined, the name element specifies a relative name, which is anchored to the name space of the name certificate issuer. If a principal is defined, the name is a fully-qualified name, which is anchored explicitly to the name space of the specified principal.
6 Other SPKI Objects

This chapter specifies briefly other SPKI objects. Because this draft version gives only the DTD definitions for these objects, readers should see [SPKI] for the actual descriptions and semantics.

6.1 ACL

<!ELEMENT acl (entry*)>
<!ELEMENT entry (subject,delegation?,tags,validity?,comment?)>

6.2 CRL and Delta-CRL

<!ELEMENT crl (version?,canceled-certs,not-before?,not-after?)>
<!ELEMENT canceled-certs (cert-hash*)>
<!ELEMENT cert-hash (hash,uri*)>

<!ELEMENT delta-crl (version?,hash-of-crl,canceled-certs,not-before?,not-after?)>
<!ELEMENT hash-of-crl (hash,uri*)>

6.3 Revalidation

<!ELEMENT reval (version?,cert-hash,not-before?,not-after?)>

6.4 One-Time

<!ELEMENT ot-reval (version?,cert-hash,one-time)>
<!ELEMENT one-time (#PCDATA)>
Tag Intersection and Java Permissions

This chapter discusses the problems of generic tag intersection. The original way to intersect SPKI authorization certificate tags (see [SPKI] section 8.3) does not fit every use. A problem in encoding and intersecting Java Permissions is demonstrated as an example.

In Java, rights to perform some critical operations can be granted by giving permissions to code. These permissions are instances of some permission class (they must be subclasses of Permission, but that is irrelevant here). The name of the permission is the name of the permission class, and can be anything. Permissions usually do have some parameters, for example a target (like the name of the file for which a FilePermission grants access). A permission can also have another set of parameters called actions (like the list of allowed access methods (read, write, execute) in a FilePermission).

One point in permissions is that anyone can create their own permissions for their own code. Because of this it is never known beforehand what kind of permissions a Java system might have. Now, if one wants to transfer these Java permissions in a secure way with SPKI certificates, a way to encode any Java permission to the tag field of an authorization certificate must be defined. The following is an example of how to accomplish that. The example uses java.io.FilePermission class:

```xml
<tag>
  <java-permissions>
    <permission>
      <name>java.io.FilePermission</name>
      <target>/etc/passwd</target>
      <action>read</action>
    </permission>
  </java-permissions>
</tag>
```

Let’s suppose that another certificate has AllPermission encoded as follows:

```xml
<tag>
  <java-permissions>
    <permission>
      <name>java.security.AllPermission</name>
    </permission>
  </java-permissions>
</tag>
```

Now, if we wanted to reduce the two certificates and had to calculate the intersection of these tags in the original way, the result would be an empty intersection of tags. However, it is obvious that the
The result of the intersection should be the FilePermission (the former permission). There would be no problems if the intersection would be calculated by comparing these permissions with the implies method of Java’s Permission interface.

The suggestion is that tags can have, in addition to normal elements and normal intersection rules, special elements which will have special intersection rules. The first of these special elements would be "java-permissions". Computing the intersection in this case would go like this:

1. Parse the Java permissions from both certificates, create one instance for each permission.
2. Test each permission of the first certificate against each permission of the second certificate, and put aside all the permissions that are implied by another. Do vice versa.
3. Remove possible duplicates from the resulting permission set.
4. Encode the permissions back to SPKI form if they are to be written into a new certificate.

The step 4 is not usually necessary, since reduced certificates are mostly kept inside a trusted memory and never communicated to anyone else.
Appendix A - The full DTD

This appendix gives an alphabetically ordered list of XML DTD declarations. The top-level objects are cert, name-cert, acl, crl, delta-crl, reval and ot-reval.

Note that although [SPKI] has been the basis for developing the XML representation of SPKI certificates, this document is not a straightforward mapping of the original BNF rules to an XML DTD. Differences between the s-expression BNF and the XML DTD defined here are due to the differences between BNF and XML DTD as languages for describing structured documents.

<!ELEMENT acl (entry*)>
<!ELEMENT canceled-certs (cert-hash*)>
<!ELEMENT cert (version?,issuer,subject,delegation?,tags,validity,comment?)>
<!ELEMENT cert-hash (hash,uri*)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT crl (version?,canceled-certs,not-before?,not-after?)>
<!ELEMENT delegation EMPTY>
<!ELEMENT delta-crl (version?,hash-of-crl,canceled-certs,not-before?,not-after?)>
<!ELEMENT dsa-g (#PCDATA)>
<!ELEMENT dsa-p (#PCDATA)>
<!ELEMENT dsa-pubkey (dsa-p,dsa-q,dsa-g,dsa-y)>
<!ELEMENT dsa-q (#PCDATA)>
<!ELEMENT dsa-y (#PCDATA)>
<!ELEMENT entry (subject,delegation?,tags,validity?,comment?)>
<!ELEMENT hash (#PCDATA)>
<!ATTLIST hash hash-alg CDATA #REQUIRED>
<!ELEMENT hash-of-crl (hash,uri*)>
<!ELEMENT hash-of-key (hash,uri*)>
<!ELEMENT issuer ((public-key|hash-of-key),uri*)>
<!ELEMENT name ((public-key|hash-of-key)?,#PCDATA)>
<!ELEMENT name-cert (version?,name-issuer,subject,valid?,comment?)>
<!ELEMENT name-issuer ((public-key|hash-of-key),name)>
<!ELEMENT new-cert (uri*)>
<!ELEMENT notafter (#PCDATA)>
<!ELEMENT notbefore (#PCDATA)>
<!ELEMENT object-hash (hash,uri*)>
<!ELEMENT one-time (#PCDATA)>
<!ELEMENT online (uri*,(public-key|hash-of-key),params)>
<!ATTLIST online type (crl|reval|one-time) #REQUIRED>
<!ELEMENT ot-reval (version?,cert-hash,one-time)>
<!ELEMENT public-key (rsa-pubkey|dsa-pubkey)>
<!ELEMENT reval (version?,cert-hash,not-before?,not-after?)>
<!ELEMENT rsa-e (#PCDATA)>
<!ELEMENT rsa-n (#PCDATA)>
<!ELEMENT rsa-pubkey (rsa-e,rsa-n)>
<!ELEMENT subject ((public-key|hash-of-key|name|object-hash|keyholder|
threshold),uri*)>
<!ELEMENT threshold (subject+)>
<!ATTLIST threshold k CDATA #REQUIRED>
<!ELEMENT uri (#PCDATA)>
<!ELEMENT validity (notbefore?,notafter?,online*,new-cert?)>
<!ELEMENT version (#PCDATA)>
Appendix B – Examples of XML-encoded Certificates

This appendix gives an example of an XML-encoded SPKI certificate. The issuer is defined as a DSA public-key and the subject is identified by a hash of a public-key. The hash is calculated from the public key of the issuer (as suggested by the URN inside the hash-of-key element). The public-key, the hash of the public-key and the signature are all valid. (The signature has been calculated by the XML Security Suite software by IBM Alphaworks.)

Note that the lines have been truncated to 72 characters. A backslash (\) has been added to the end of every truncated line. The rest of the truncated line continues in the next line and is indented in the same way as the previous one.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/01/xmldsig/">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/1999/WD-xml-c14n-19991115"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/01/xmldsig/dsa"/>
    <Reference IDREF="Res0">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/TR/1999/WD-xml-c14n-19991115"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/01/xmldsig/sha1"/>
      <DigestValue Encoding="http://www.w3.org/2000/01/xmldsig/base64">
        oYh7FFSLnv1C8OUiNTZQOdQOdQmQ=
      </DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>
    MCwCFHWh6RHPga3j30VPFFaEYcZSBAAAhRnyTvffCDVB068BX2reKwhTN1kSA==
  </SignatureValue>
  <dsig:Object Id="Res0" xmlns="" xmlns:dsig="http://www.w3.org/2000/01/xmldsig/">
    <cert>
      <version>0</version>
      <issuer/>
      <public-key>
        <dsa-pubkey>
          AP1/U4EddRlpUt9Knc7s5Of2EbdSP09EAMMeP4C2USZpRV1A1\1H7WTZWNPq/xfW6MPbLm1Vs14E7gB0b/JmYldrmVC1pJ+f6AR7ECLCT\7up1/63xhv401fmxqimFQ8E+4P208UewwIllVBNaFpEy9nXzrithlyrv8\1IDGZ3RSAH
```

Paajarvi [Page 27]
<dsa-q>AJdgUI8VIwVMspK5qgLrhAvwWBz1</dsa-q>
<dsa-g>APfhoIXWmz3ey7yrXDa4V7151K+7+jrqgv1XTAs9B4JnUV1Xj\rrUWU/mtQCqCq1C08SRZxI+hMKBYTt88JMozipuE8FnqLVHyNKOCjrh4rs\6Z1kW6jfTw61TVi8ftiegEkoO8yk8b60UZCJqlf4Vrlnwa5i2ZegHtVJ\WQBTDv+z0kq</dsa-g>
<dsa-y>e45XbC1Klny81WIBJi3uX46+fzbYjt6jiApSgoFvVZvtTFOB\ekft18ccJwcm8c8PCtHJkiPDwavQLzAwymMsKYoj3DvozzECIyaQRi\scfn4pimJZXWqSusxRydBFf9atAhpZeWFjqTGM0CLmDj5Uetc6wNq\8plJXZe8AM==</dsa-y>
</dsa-pubkey>
</public-key>
</issuer>
<hash-of-key>
<hash hash-alg="sha1">AMmGTeQjk65b82Jggdp+0A5MOMO=</hash>
<uri>urn:spki:issuer</uri>
</hash-of-key>
</subject>
<delegation/>
<tag>
<java-permissions>
<permission>
<type>java.io.FilePermission</type>
<target>/examples/*</target>
<action>read</action>
</permission>
<permission>
<type>java.security.SecurityPermission</type>
<target>setPolicy</target>
</permission>
</java-permissions>
</tag>
</validity>
<notbefore>2000-02-21_12:00:00</notbefore>
<notafter>2000-08-31_24:00:00</notafter>
</validity>
<comment>An example of XML-encoded SPKI certificate.</comment>
</cert>
</dsig:Object>
</Signature>
References


Acknowledgements

I want to thank many people for contributing to the work of creating this Internet-draft. The first one to thank is Petteri Koponen who came up with the idea, presented it to me and suggested me to write this Internet-draft. Juha Koponen has been of most help during the writing process. His feedback has been really valuable. I also thank Heli Harri for proof-reading and making corrections. Juhana Rasanen, Pekka Nikander, Jonna Sars and Harri Jaalinoja have all been of great help especially in the early phases of the work. I would also like to thank Tatu Ylonen for advice and help in the last meters. I also thank Carl Ellison for his insightful comments.

Author’s Address

Juha Paajarvi
First Hop Ltd.
Tekniikantie 12
FIN-02150 ESPOO
Finland

E-mail: juha.paajarvi@firsthop.com

Expiration and File Name

This draft expires in September 2000.

The file name of this draft is draft-paajarvi-xml-spki-cert-00.txt