Simple Public Key Certificate

INTERNET-DRAFT

Expires: 18 September 1998

Bill Frantz
Electric Communities

Butler Lampson
Microsoft

Ron Rivest
MIT Laboratory for Computer Science

Brian M. Thomas
Southwestern Bell

Tatu Ylonen
SSH

13 March 1998

Simple Public Key Certificate

<draft-ietf-spki-cert-structure-05.txt>

Status of This Document

This document supersedes the draft filed under the name draft-ietf-spki-cert-structure-04.txt.

This version introduces "rsa-pkcs1" as one option for <pub-sig-alg-id>, while the working group considers the question of the proper place to bind hash algorithm choice. It specifies the <sig-val> structure needed by that option.

This version has removed the secret-key definitions, as requested at the meeting in December 1997.

The theory behind this kind of certificate is to be found in draft-ietf-spki-cert-theory-*.txt. Examples of certificate uses are to be found in draft-ietf-spki-cert-examples-*.txt. The requirements behind this work are listed in draft-ietf-cert-req-*.txt.

Distribution of this document is unlimited. Comments should be sent to the SPKI (Simple Public Key Infrastructure) Working Group mailing list <spki@c2.net> or to the authors.

This document is an Internet-Draft. 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. Internet-Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet-Drafts as reference material or to cite them other than as a "working draft" or "work in progress."

To learn the current status of any Internet-Draft, please check the lid-abstracts.txt listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (East USA), ftp.isi.edu (West USA), nic.nordu.net (North Europe), ftp.nis.garr.it (South Europe), munnari.oz.au (Pacific Rim), or ftp.is.co.za (Africa).

Abstract

This document specifies a standard form for digital certificates and access control lists. These structures bind either names or authorizations to keys or names that resolve to keys. The name and authorization structures can be used separately or together. We use S-expressions as the standard format for these certificates and define a canonical form for those S-expressions.

These structures are also known under the name SDSI 2.0.
Table of Contents

Status of This Document........................................1
Abstract..................................................................2

Table of Contents...............................................3

1. Overview of Contents.......................................5

2. Glossary.....................................................6

3. Primitives...................................................8
   3.1 Canonical S-expression................................8
   3.2 <byte-string>.........................................8
   3.3 S-expression..........................................9
   3.4 Encoding examples...................................9
   3.5 Use of canonical S-expressions....................10
   3.6 Advanced S-expressions.............................10
   3.7 Unique IDs...........................................11
   3.8 Primitive Objects.................................11
      3.8.1 <pub-key>......................................12
      3.8.2 <hash>..........................................14
      3.8.3 <signature>....................................14
      3.8.3.1 <sig-val>..................................14

4. Authorization Certificate.................................16
   4.1 <version>............................................16
   4.2 <cert-display>.......................................16
   4.3 <issuer>.............................................17
   4.4 <issuer-loc>.........................................17
   4.5 <subject>............................................17
   4.5.1 <obj-hash>.......................................18
   4.5.2 <keyholder>......................................18
   4.5.3 <subj-thresh>....................................18
   4.6 <subject-loc>........................................19
   4.7 <deleg>...............................................20
   4.8 <tag>................................................20
   4.9 <valid>.............................................20
   4.9.1 <date>............................................21
   4.9.2 <online-test>....................................21
   4.10 <comment>..........................................22

5. Name certificate.............................................23
   5.1 Name certificate syntax...........................23
   5.2 <name>..............................................24
   5.3 Name reduction......................................24

6. ACL and Sequence formats...............................26
   6.1 <acl>................................................26
   6.2 <sequence>..........................................27

7. On-line test reply formats...............................28
7.1 CRL and delta-CRL...........................................28
7.2 Revalidation................................................28
7.3 One-time revalidation.................................29

8. 5-Tuple Reduction...........................................30
8.1 <5-tuple> BNF...........................................30
8.2 Top level reduction rule...............................31
8.3 Intersection of tag sets..............................31
8.4 Reduction of (subject (threshold ..))................32
8.7 Certificate Result Certificates.......................32

9. Full BNF...................................................34
9.1 Top Level Objects......................................34
9.2 Alphabetical List of BNF Rules.......................34

References..................................................37

Acknowledgments............................................39
Authors’ Addresses..........................................39
Expiration and File Name.................................40
1. Overview of Contents

This document contains the following sections:

Section 1: this overview.

Section 2: a glossary of terms.

Section 3: the definition of structure primitives used throughout the rest of the document.

Section 4: the definition of an authorization certificate and its component parts.

Section 5: the definition of a name certificate and the few parts that differ from an authorization certificate.

Section 6: the definition of an ACL and a (sequence...) structure.

Section 7: the definition of on-line test reply formats. An on-line test is a mechanism for asking for a CRL or a revalidation. The replies are CRLs or revalidations.

Section 8: the rules of 5-tuple reduction

Section 9: the full BNF.

The References section lists all documents referred to in the text as well as readings which might be of interest to anyone reading on this topic.

The Acknowledgements section.

The Authors’ Addresses section gives the addresses, telephone numbers and e-mail addresses of the authors.
2. Glossary

We use some terms in the body of this document in ways specific to SPKI:

5-TUPLE: The 5 security-relevant fields from a certificate or ACL entry, sometimes abbreviated <I,S,D,A,V>. [See "certificate", below.]

ACL: Access Control List -- a list of entries binding some attribute to an identified entity. For our purposes, an ACL entry is like a certificate, except that it is "issued" by "self" and need not be signed. It yields a 5-tuple of the form <self,S,D,A,V>.

CERTIFICATE: a digitally signed record binding one or more attributes to a global identifier or to a name that can be resolved to a global identifier. The certificate is assumed to have up to 5 kinds of field with security value: Issuer, Subject, Delegation permission, Authorization, Validity dates and/or tests.

CANONICAL S-EXPRESSION: an encoding of an S-expression that removes options and is designed for easy parsing.

KEYHOLDER: the person or other entity that owns and controls a given private key is said to be the keyholder of the corresponding public key.

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

NAME: a SDSI name always relative to the definer of some name space. This is sometimes also referred to as a local name. A global name includes the global identifier of the definer of the name space. For example, if

    (name jim)

is a local name,

    (name (hash md5 \+gbUgUltGysNgewRwu/3hQ==\) jim)

could be the corresponding global name.

ON-LINE 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.

PRINCIPAL: a signature key, capable of generating a digital signature.

PROVER: the entity that wishes access or that digitally signs a document.
SPEAKING: a Principal is said to "speak" by means of a digital signature. The statement made is the signed object (typically a certificate, for SPKI purposes).

S-EXPRESSION: the data format chosen for SPKI/SDSI. This is a LISP-like parenthesized expression with the limitations that empty lists are not allowed and the first element in any S-expression must be a string, called the "type" of the expression.

VALIDITY CONDITIONS: a date range that must include the current time and/or a set of online tests that must succeed before a certificate or ACL entry is to be considered valid.

VERIFIER: the entity that processes requests from a prover, including certificates. The verifier uses its own ACL entries and certificates provided by the prover to perform "5-tuple reduction", to arrive at a 5-tuple it believes about the prover: <self,prover,D,A,V>.
3. Primitives

We have chosen a simplified form of S-expression (the canonical form) as the format for SPKI objects. An S-expression is a list enclosed in matching "(" and ")". We assume the S-expression technology of [SEXP] with the restrictions that no empty lists are allowed and that each list must have a byte string as its first element. That first element is the "type" or "name" of the object represented by the list.

SPKI objects are defined below in a familiar extension of BNF -- with "|" meaning logical OR, "*" meaning closure (0 or more occurrences), "?" meaning optional (0 or 1 occurrence) and "+" meaning non-empty closure (1 or more occurrences). A quoted string represents those characters. First we define the canonical S-expression form in that BNF.

For the sake of readability, all examples and the BNF in this document specify advanced rather than canonical S-expressions. That is, single word strings that start with alphabetic characters are used without quotes and strings can be in hex, base64 or double-quoted ASCII. The mapping to canonical form is specified below.

3.1 Canonical S-expression

We define a canonical S-expression as containing binary byte strings, each with a given length, and punctuation "()[]" for forming lists. The length of a byte string is a non-negative ASCII decimal number, with no unnecessary leading "0" digits, terminated by ":". We further require that there be no empty lists and that the first list element be a byte string (as defined below). This form is a unique representation of an S-expression and is used as the input to all hash and signature functions. If canonical S-expressions need to be transmitted over a 7-bit channel, there is a form defined for base64 encoding them.

3.2 <byte-string>

A byte string is a binary sequence of bytes (octets), optionally modified by a display type.

If the byte-string is used as a binary integer, these bytes are twos-complement, in network standard order (most significant byte first). It is up to the application whether these are considered signed or unsigned.
All byte strings carry explicit lengths and are therefore not 0-terminated as in the C language. They are treated as binary even when they are ASCII, and can use any character set encoding desired. Typically, such a choice of character set would be indicated by a display type.

A display type is assumed to be a MIME type giving optional instructions to any program wishing to display or use the byte string. For example, it might indicate that the string is in UNICODE, is a GIF or JPEG image, is an audio segment, etc. Although the display type of a byte string is optional, it is considered part of the string for any equality comparisons or hashing. That is, two strings of the same bytes will not be considered equal if they have unequal display types.

A byte-string is defined by:

\[
<\text{byte-string}>:: <\text{bytes}> | <\text{display-type}> <\text{bytes}> ;
\]

\[
<\text{bytes}>:: <\text{decimal}>":" (binary byte string of that length) ;
\]

\[
<\text{decimal}>:: <\text{nzddigit}> <\text{ddigit}>* | "0" ;
\]

\[
<\text{nzddigit}>:: "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
\]

\[
<\text{ddigit}>:: "0" | <\text{nzddigit}> ;
\]

\[
<\text{display-type}>:: "[" <\text{bytes}> "]" ;
\]

### 3.3 S-expression

An S-expression is of the form:

\[
<\text{s-expr}>:: "(" <\text{byte-string} > <\text{s-part}>* ")" ;
\]

\[
<\text{s-part}>:: <\text{byte-string}> | <\text{s-expr}> ;
\]

where the first byte string in the S-expression is referred to here as its "type".

### 3.4 Encoding examples

\[
(4:test26:abcdefhijklmnopqrstuvwxyz5:123455::: ::)
\]

is a canonical S-expression consisting of four byte strings: "test", "abcdefhijklmnopqrstuvwxyz", "12345" and ":: ::".

The advanced text form is:

\[
(\text{test abcdefhijklmnopqrstuvwxyz } 12345 ":: ::")
\]
showing that the advanced form follows familiar token recognition rules, not permitting tokens to start with digits, terminating them with white space or punctuation marks.

For transmission of true 8-bit forms, we permit base64 encodings according to [RFC2045], with the base64 characters enclosed in braces. The example above encodes to:

```
{KDQ6dGVzdD12OmFiY2RlZmdoaWprbG1ub3cnN0dXZ3eHl6NToxMjM0NTU6OjogOjop}
```

### 3.5 Use of canonical S-expressions

Canonical S-expressions were designed to be as simple to pack and parse as possible. Some concessions were made to those developers who might want to examine a canonical S-expression in an ASCII editor like emacs (specifically the readable decimal length fields and readable "()[]" characters) but in general the form is as close to minimum size as possible. Parsing of a canonical form S-expression requires minimal look-ahead and no re-scanning of incoming bytes. As a result, the parsing code remains very small. Assuming each byte string is stored with a length field, generation of a canonical form from a data structure requires an extremely small amount of code.

The canonical S-expression is the form which is hashed for both generating and verifying signatures. These two processes can be thought of as the start and end of an SPKI object’s useful life and both require canonical form. Therefore, it is recommended that the canonical form be the form transmitted and stored in normal use, to be converted temporarily to and from a more readable form by display or editing applications written for the purpose.

[Violating that suggestion, this document includes some advanced forms for readability. Since this document is required to be straight ASCII, no pure 8-bit canonical forms will be presented except under base64 encoding.]

### 3.6 Advanced S-expressions

[SEXP] includes a general purpose utility program for converting between canonical and advanced S-expression form. In the advanced form, individual byte strings may be expressed without length fields (if they are what most languages consider text tokens), may be written as quoted strings (under normal C string rules), or may be individually hex or base64 encoded. Also in the advanced form, white space between list elements is allowed for readability and ignored on
conversion to canonical form.

For examples, this document will normally use the advanced form because of its readability, but for at least one concrete example the canonical form and its hash are presented (base64 encoded where necessary, given that this document is 7-bit ASCII).

In these examples, we will use keywords without preceding length fields, quoted strings, hex values (delimited by "#") and base64 values (delimited by "|"). Those are features of the advanced transport form of an S-expression, and are not part of the canonical form. We will always present the canonical form (base-64 encoded, when it contains non-ASCII characters) which the reader can decode to get the actual canonical form.

3.7 Unique IDs

Top level object names are defined in this document along with certain algorithm names. <tag> objects are user-defined, using a language for describing sets of permissions given here, and in the process, the defining user can choose any object names he or she wishes.

For the definition of new algorithm names, it is our preference that this be taken on by IANA [RFC1780] for single-word standard names.

In the interest of maximum flexibility we also permit users to define their own algorithm names via a normal URIs (which presumably point to descriptions of the algorithms or even to code).

3.8 Primitive Objects

The objects defined in SPKI/SDSI 2.0 are S-expressions. That is they are lists of either byte strings or other lists. In our case, all S-expressions start with a <byte-string>, called the object name. The remaining elements of the list are called "parts" of the object.

In a communication from prover to verifier, one might encounter only a small number of different objects: usually a <sequence> of <cert>, <pub-key>, <signature> and <op>. The verifier will also need to refer to its own <acl>. These are considered top level objects and are defined in the sections immediately following.

It is standard SPKI/SDSI practice to use names starting with a lower case letter, followed by lower case letters, digits and hyphens for object types. SPKI/SDSI is case-sensitive, so the byte-string "RSA"
is not the same as "rsa". Non-standard object types (i.e. <tag>s
defined by an application developer) are unconstrained, may have
display types and may even be URIs pointing to documentation of the
object type.

The structure and interpretation of the parts is up to the designer
of the top-level object type. However, for the sake of
simplification, we have decided that all objects are "positional".
That is, their parts are listed in some fixed order with meaning of
the part depending on its position. Parts can be omitted only by
omitting a contiguous set of trailing parts. Exceptions to this are
found in the top level <cert> and <acl> constructs.

The following are the definitions of the top level objects which a
verifying program may encounter. Note that the main object, <cert>,
is sub-type based so the parameter fields may be in any order, but
the BNF suggests a fixed order. We use the BNF definition to
indicate that there may not be more than one of each of the listed
fields, and also to suggest (for readability) that the certificate
parts be presented in the order given. This document will use that
order.

3.8.1 <pub-key>

<pub-key>:: "(" "public-key" "(" <pub-sig-alg-id> <s-expr>* ")"
<uris> ")" ;

A public key definition gives everything the user needs to employ the
key for checking signatures. The <uri>s, if present, give locations
where one might find certificates empowering that public key.

The only pub-sig-alg-id’s we have defined at this point are for
signature verification. That is because we need only signature keys
for certificate formation and access control. Other key types are
open to being defined by users.

The following is an RSA signature key, shown in advanced transport
format:

(public-key
 (rsa-pkcs1-md5
  (e #03#)
  (n |ANHCG85jXFGmicr3MGpj53FYYSY1aWAue6PKnpFErHhKMJa4HrK4WSKTO
  YTTlapRzn9ELD2D71Wd3Q8PD01yi1NjP20NzKxQVHrrAniQoczeO2uiz/yY
  VDzJ1djImixyb/Jyme3D000UXhd6VGAz0x0cxgrKefKnmjy41DKro3uW1| ))

For actual use, the key is held and presented in canonical form the
base64 encoding of which is:

```
[KDEwOnB1YmxpYy1rZXkoMTM6cnNhLXBrY3MxLWlkNSgxOmUxOgMpKDE6bJEyOToA0cIbfzmnCUuAajyvchl+PncVhjyjvPC5708eqkuUSeEowlrgrshR2IPm
ShNQvq1H0ecQsPSYyPuVZidDw8PSXKL00mk3MyTFBuesCchChzN45m6LP/JhU
PMnUNz1iaLHjv8nK27cFRSJRJeFpUyD7TRycSp58qeaPLjXQquje5BuKPks=]
```

Although not strictly needed by this draft, the private key for the public key above is:

```
(private-key
  (rsa-pkcs1-md5
    (e #03#)
    (n |ANHCG85jXFGmicr3MGpj53FYYSY1aWAue6PKnpFERHHJKMa4HRK4WSKT
      OTY11apRznnELD2D71Wd3Q8PD0lyi11NjPnZmxQVHrrAnIQoczeO2ui/
yVdzJ1DdiImiyb/Jyme3DOiUXhd6VGAz0x0cgrKeFkmmjy410Kro3u
      Wl|)
    (d |AIvWvTRCPYyEW9ykuy1CmkvQMQm5V0Um0xvwuDHawGywy8lacx65hcM
      Q0M3uXw2laacYcKcnu0+k19fX42MX0D7cLN/Qrql8F6x5mczoG+EpO6F
      F+cvgXfupelVM6PmJdIiaJrERThU01Pr12N+NnAL7CvU6X1nhOfn/277
      iz|)
    (p |APesjZ8gK4RGV5Qs1eCRAvP7mVblgf13R5fwApw6bTVWzunIwk/2sSh
      ytpc90edr+0DPwlndnvEXTUY1df0DwPc=|)
    (q |ANjPQe600JfV90GWE3q2c9724AX7FKx64g2F81xgiWWQKEeq1WiiEDx
      7qh01LrhmBT+VXEDFRG2LHmuNSTzj7M=|)
    (a |AKUds79qs62E0mLIjpW2AO0b9EOZAVOK2mVKrGgm83jkifEwgYqkdhr3
      MebopNpH/NXfluTv0tk3i70TqitK08=|)
    (b |AJCKK/RFNnbfq+iu5Y1H09+n56q6nYx2nQV5ZTD2VsO54KxYUcW5sWtX2
      nxr4YydBEA3+46CsuLZ5cvvJeMNNnc=|)
    (c |CIPwAAO8Vmj0/BfCtsg+35+r94jxwYGH263RsqyNxbvkAO6xP8ht8/v
      zdr93eX5B9ZKBQg1HWHCsbQbQtmN=|))
```

or

```
[KDExOnBya2XhgdGuta2V5KDE3OnJzYs1wa2NzNS1tZDUoMTplMToDKGx5o4
xMj56ANHCG85jXFGmicr3MGpj53FYYSY1aWAue6PKnpFERHHJKMa4HRK4WSK
TOY11apRznnELD2D71Wd3Q8PD0lyi11NjPnZmxQVHrrAnIQoczeO2ui/
YVdzJ1DdiImiyb/Jyme3DOiUXhd6VGAz0x0cgrKeFkmmjy410Kro3u
Wl|)
```

Ellison, et al. [Page 13]
where a, b and c are CRT parameters.

3.8.2 <hash>

A <hash> object gives the hash of some other object. For example, the public key given above has the following hashes:

(hash md5 #9710f155723bc5f4e0422ea53ff7c495#)
{KDQ6aGFzaDM6bWQ1MTY6lxDxVXI7xfTgQi6lP/fElSk=}

(hash sha1 #1a6f6d62 1abd4476 f16d0800 fe4c32d0 6ff62e93#)
{KDQ6aGFzaDM6bWQ1MTY6lxDxVXI7xfTgQi6lP/fElSk=}

3.8.3 <signature>

A signature object is typically used for a certificate body and typically follows that <cert> object in a <sequence>. One can also sign objects other than certificate bodies, of course. For example, one can form the signature of a file.

3.8.3.1 <sig-val>

A <sig-val> object is typically used for the algorithm listed in the public key.

For rsa-pkcs1-md5 and rsa-pkcs1-sha1, <sig-val> is a <byte-string> -- the value of the RSA signature operation.

For dsa-sha1, <sig-val> is a <byte-string>, consisting of the concatenation of the values r and s (in that order) from the DSA. Each is of the length of the sub-prime, q. We could split these
values out in an S-expression, but at least one popular cryptographic package (BSAFE) assumes the two values are concatenated so that splitting and recombining would be extra work for the programmer.

For rsa-pkcs1 (should that option be preferred by the working group over the specification of hash algorithm in the <pub-sig-alg-id>), <sig-val> would need to be:

<sig-val>:: "(" "rsa-pkcs1-sig" <hash-alg-name> <byte-string> ")" ;

Custom algorithms, specified by URI, might need custom <sig-val> definitions. The <sig-val> structure for a custom <pub-sig-alg-id> should be specified at the given URI even if it is one used by other algorithms.
4. Authorization Certificate


The basic certificate form is an authorization certificate. It transfers some specific authorization or permission from one principal to another. The fields defined here assume one wants SPKI certificates without SDSI name definition. Some of those field definitions are modified in Section 5, to provide name definition.

Because a certificate merely transfers authorizations, rather than creating them, the form we call ACL-entry is also defined below to inject authorizations into a chain of certificates. An ACL entry lives on the machine of the verifier, leading to the observation that all authorization flow is in a circuit -- from the verifying machine’s ACL, possibly through certificates and then back to the verifying machine. Alternatively, one might say that the only root of an authorization certificate chain is the verifier.

4.1 <version>

<version>:: "(" "version" <byte-string> ")" ;

Version numbers are alphanumeric strings. If the <version> field is missing from an object, it is assumed to be (version "0"), which is the version of all objects in this draft. Elaboration of version numbers, possibly with multiple fields, are left for later to define.

A certificate containing an unrecognized version number must be ignored.

4.2 <cert-display>

<cert-display>:: "(" "display" <byte-string> ")" ;

This optional field gives a display hint for the entire certificate. This display parameter does not affect certificate chain reduction, but is provided to aid user-interface software in certificate display.

At this time, we have no such hints defined. This field is up to developers to define as they see fit. For verifiers of certificates, this field is treated as a comment.
4.3 <issuer>

<issuer>:: "(" "issuer" <principal> ")" ;

<principal>:: <pub-key> | <hash-of-key> ;

<hash-of-key> might be the preferred <principal>, not merely for size but also in case one is using small RSA keys and protecting them from cryptanalysis by keeping them secret.

4.4 <issuer-loc>

<issuer-loc>:: "(" "issuer-info" <uris> ")" ;

The (issuer-info) object provides the location of the certificate(s) by which the issuer derives the authority to pass along the authorization in the present <cert>. We expect the prover (the calling client) to track down such other certificates and provide them to the verifier (the called server), but we allow this information in the certificate to simplify that process for the prover.

4.5 <subject>

<subject>:: "(" "subject" <subj-obj> ")" ;

<subj-obj>:: <principal> | <name> | <obj-hash> | <keyholder> | <subj-thresh> ;

In the most basic form,

<subj-obj>:: <principal> ;

and one may make an SPKI implementation with only that definition, in case names are considered unnecessary for the intended application.

However in full-blown implementations, the subject may also be a name, representing a group of principals or a delayed binding to some one principal, the hash of an object, or a K-of-N threshold of principals (in which case, the authorization being granted to the subject is being spread out among multiple parties that must cooperate to exercise that authorization). The <keyholder> case is special and of little interest to verifier code, since it is used in a certificate that is a message to a human.

See section 5 for the definition of <name>.
4.5.1 <obj-hash>

<obj-hash>:: "(" "object-hash" <hash> ")" ;

This option for a (subject ) refers to an object other than a <principal>. One might use this form to assign attributes to an object (a file, a web page, an executable program, ...).

4.5.2 <keyholder>

<keyholder>:: "(" "keyholder" <keyholder-obj> ")" ;
<keyholder-obj>:: <principal> | <name> ;

This form of subject refers to the flesh and blood (or iron and silicon) holder of the referenced key. A <cert> with such a subject is saying something about that person or machine -- such as its location, its address, its age, its weight, its height, its picture, .... Such a certificate is most probably a message to a human rather than for use in a verification process, but we anticipate applications that will appreciate the machine-readable format of such information.

4.5.3 <subj-thresh>

<subj-thresh>:: "(" "k-of-n" <k-val> <n-val> <subj-obj>* ")" ;

where K < N, and there are N <subj-obj> subjects listed.

A threshold subject, introduced by Tatu Ylonen for SPKI and by Rivest and Lampson in SDSI 1.0, specifies N subjects for a certificate or ACL entry, of which K must agree before the permission is passed along.

The actual intent is to insure that there are K distinct paths passing permission between the verifier’s ACL and the prover’s request. These multiple paths fork and join, so the k-of-n construct could theoretically be part of either the Subject or the Issuer. Since an ACL might want to specify these multiple paths (and an ACL has no Issuer) and since a certificate is signed by a single Issuer, we have chosen to specify the branching at the Subject.

A certificate or ACL with a k-of-n Subject does not delegate permission to any of those subjects, alone. Rather, each of these subjects receives a share of the delegated permission. Only if at least K of the N subjects show certificate paths which converge on a single target Subject during reduction, is that permission
transmitted to the target. If fewer than K such paths can be shown, then the permission is not delegated.

This construct is far from simple. However, it is extremely useful. It has been demanded by a number of initial customers of SPKI certificates. It also solves a number of sticky political problems. This section lays out the specification of K-of-N subjects. The rules for reducing 5-tuples containing such entries are given later.

Examples of the use of K-of-N permission propagation include:

1. co-signing of electronic corporate checks or purchase orders above a certain amount

2. establishing the root DNSSEC key, bypassing the political battles which would inevitably ensue if one country were to hold *the* root key for the entire world. The same goes for any root key.

3. establishing a root key for a trusted service, via multiple algorithms. That is, one could have three root keys, using RSA, DSA and Elliptic Curve signature algorithms (for example), and require that two of them yield a valid chain. This way, if someone were to break an entire algorithm (find a way to invert the algorithm), much less if someone were to break one key in the set of three, the root remains securely established. At the same time, there is fault tolerance. In case one of the keys is revoked, the following certificates remain empowered.

4. using online and off-line issuers. One could have a permission established by an off-line key issuing a long-lived certificate and echoed by an online automated server, issuing short-lived certificates. The delegation of this permission could require both before the eventual subject gets the permission. This can be achieved through the use of (online ) tests in a long-lived certificate, but the K-of-N subject mechanism may be cleaner.

5. ultra-secure applications. There are many applications which follow the nuclear weapons launch scenario. That is, multiple agreement is required before the permission is granted.

4.6 <subject-loc>

<subject-loc>:: "(" "subject-info" <uris> ")" ;

This optional field provides the location of information about the subject. For example, if the subject is a hash of a key, this might provide the location of the key being hashed. If the subject is a SDSI name, it might give the location of a SDSI name certificate.
server.

4.7 <deleg>

<deleg>:: "(" "propagate" ")" ;

This optional field, if present, notes that the <subject> has not only the permission given in the <cert>’s <tag> field but also the permission to delegate that (or some portion of it) to others.

4.8 <tag>

<tag>:: "(" "tag" "(*)" ")" | "(" "tag" <tag-expr> ")" ;

The form "(tag (*))" means "all permissions".

The simplest tag is an S-expression with no *-forms. This is a specific permission which must be passed along and used intact.

A tag with *-forms represents a set of specific permissions. Any subset of such a set of permissions may be delegated by a principal empowered to delegate. When one is reducing the 5-tuples from such certificates, one intersects the adjacent tag sets to find a resulting tag set.

All tags are assumed to be positional. That is, parameters in a tag have a meaning defined by their position.

All tags are assumed to be extendable. That is, if one adds a field to the end of a tag definition, one is restricting the permission granted. [If the field added makes the tag invalid, then one has restricted the original permission to zero.]

See the full BNF section for the full tag body BNF, including specification of *-forms.

4.9 <valid>

The <valid> field gives validity dates and/or on-line test information for the certificate.

<valid>:: <not-before>? <not-after>? <online-test>* ;

<not-after>:: "(" "not-after" <date> ")" ;
The not-after and not-before options are self-explanatory. If either is missing, then the certificate is assumed valid for all time in that direction. For example, one might omit the <not-before> field, if that date would be before or at the time of creation of the certificate, unless one wanted to note the creation time for documentation purposes.

4.9.1 <date>

A date field is an ASCII byte string of the form:

YYYY-MM-DD_HH:MM:SS

always UTC. For internal use, it is treated as a normal byte string. For example, "1997-07-26_23:15:10" is a valid date. So is "2001-01-01_00:00:00". <date> fields are compared as normal ASCII byte strings since one never needs to compute the size of a time interval to test validity -- only determine greater-than, less-than or equal.

4.9.2 <online-test>

The online test option allows a certificate to be backed up by finer grain validity testing. The reply from an online test is a digitally signed object, validated by the <principal> given in the test specification. That object includes validity dates, so that once one has the online test response, its validity dates can be intersected with the parent certificate’s validity dates to yield the current working validity dates for the certificate.

The crl form tells the verifier (or prover, who fetches this information for the verifier, in our standard model), the current list of invalid certificates. If the present certificate is not on that list, then the certificate is presumed valid.

The re-validate form is the logical opposite of the crl. It tells the verifier a list of valid certificates or, more likely, just that
the current certificate is valid.

The one-time form is a re-validate form without validity dates. It must be fetched by the verifier, rather than the prover, since it is valid only for the current verification step. [In effect, it has a validity period of just "now".]. The process of getting this one-time revalidation involves sending a unique (and partly random) challenge which is returned as part of the signed response.

If there are multiple URIs specified, any one of them can be used.

If the URI specifies an HTTP connection to the on-line test, then that URI can provide all parameters needed (e.g., a hash of the certificate in question), but in other cases, one might need to list such parameters in the optional <s-part>s.

See section 7 for a full description of on-line test reply formats.

4.10 <comment>

<comment>::  "(" "comment" <byte-string> ")" ;

This optional field allows the issuer to attach comments meant to be ignored by any processing code but presumably to be read by a human.
5. Name certificate

Names are defined for human convenience. For actual trust engine computations, names must be reduced to keys. This section gives the form of a name, a name certificate and the rules for reducing name certificates to simple mappings from name to key.

Note that we do not include an <issuer-loc> option for a name certificate. The issuer needs no authorization in order to create names. Every issuer has that right.

Similarly, there is no "certification practice statement" for these name certificates. Nothing is implied by a name certificate about the principal(s) being named. A name can be an arbitrary byte string assigned by the issuer and is intended to be meaningful only to that issuer, although other parties may end up using it. A name is not required or expected necessarily to conform to any name string in the physical world or in any other issuer’s name space.

That said, it is possible to map name certificates generated by a commercial Certification Authority into SDSI names and thus refer to keys defined under that process from within SPKI/SDSI certificates.

5.1 Name certificate syntax

A name certificate has the form:

(cert
  (issuer (name <principal> <name>))
  <subject>
  <valid>
)

<name-cert>:: "(" "cert" <version>? <cert-display>? <issuer-name> <subject> <valid> <comment>? ")" ;

<issuer-name>:: "(" "issuer" "(" "name" <principal> <byte-string> ")")" ;

That form maps directly into the intermediate form needed for name string reduction. The name must be under the <principal> of the certificate issuer, and under this syntax the certificate issuer <principal> is taken from the (name..) structure.

In a name certificate, the (tag) field is omitted and (tag (*)) is assumed. There is also no <deleg> field. A name definition is like an extension cord, passing everything the name is granted through to the subject.
The subject is unrestricted. It is what you are trying to name.

If there is more than one name certificate for a given name, with different subjects, then that name is a group. More specifically, all name certificates define groups, many of which will have only one member. A multi-member group is like a multi-plug extension cord, passing everything the name is granted through to any and all of its subjects.

5.2 <name>

The <name> form is a option for <subject>, when one wants to generate a certificate granting authorization to either a named group of principals or to a principal that has not been defined yet. This can be either a relative name or a fully-qualified name.

\[
\text{<name>:: } \text{<relative-name> | <fq-name> ;}
\]

\[
\text{<relative-name>:: \(" \text{name} \text{<names> \"})\" ;}
\]

\[
\text{<fq-name>:: \(" \text{name} \text{<principal> <names> \"})\" ;}
\]

\[
\text{<names>:: <byte-string>+ ;}
\]

A relative name is defined only with respect to an issuer and should show up only in a certificate, borrowing the <principal> from the issuer of that certificate. For evaluation purposes, the relative name is translated into a fully-qualified name before reduction.

Unlike the <issuer-name>, which is forced to be a name in the issuer’s name space, the subject name can be in any name space.

5.3 Name reduction

Given the name definition

\[
\text{(cert (issuer (name (hash md5 |Txoz1GxK/uBvJbx3prIhEw==|) fred)) (subject (hash md5 |25pxCD64YwgS1IY4Rh61oA==|)) (not-after "2001-01-01_00:00:00"))}
\]

\[
(KDQ6Y2VydCg2Om1zc3V1cig0Om5hbWUoNDpoYXNoMzptZDUxNjpPGjPUB6Er +4G81vHemiETKTQ6ZnJlZCkpR0C63V1iamVjdCg0Omhhc29zOm1kNTE2OmacQg+uGMIEtSGOEYetaApK5q5Om5vdC1hZnRlcjE5OjIwMDEtMDEtMDFfMDA6MDA6MDA6pKQ==)}
\]
the name

(subject (name (hash md5 |Txoz1GxK/uBvJbx3prIhEw==|) fred sam george mary))

reduces to

(subject (name (hash md5 |Z5pxCD64YwgS1IY4Rh61oA==|) sam george mary))

recurrung until the name reduces to a principal.  In non-pathological cases this is the only reduction rule needed.

It is possible for someone to generate a trouble-making name certificate, such as:

(cert
  (issuer
    (name (hash md5 |Txoz1GxK/uBvJbx3prIhEw==|) fred))
  (subject (name fred sam))
  (not-after "2001-01-01_00:00:00"))

in which case the reduction would grow without bound. Pairs of principals could conspire to produce loops of name definition. Therefore, the name reduction code needs to do loop detection.
6. ACL and Sequence formats

ACL and sequence structures are in the grey area. ACLs are private to one developer or application. Sequences can be thought of as part of the protocol using certificates.

6.1 <acl>

```
<acl>:: ": \(" acl \"version\)? <acl-entry>* \"\"");

<acl-entry>:: ": \(" entry \"subj-obj\) <deleg>? <tag> <valid>? <comment>? \"\"");
```

An ACL is a list of assertions: certificate bodies which don’t need issuer fields or signatures because they are being held in secure memory. Since the fields of the ACL are fields of a <cert>, we will not repeat those common field definitions here. Since an ACL is not communicated to others, developers are free to choose their own formats.

If all the optional fields are left out, the subject is given the permission specified in <tag>, without permission to delegate it, with no expiration date or condition (until the ACL is edited to remove the permission).

For example:

```
(acl
  (entry
    (name (hash md5 |plisZirSN3CBscfNQSbiDA==|) sysadmin/operators)
    (tag (ftp db.acme.com root)))
  (entry
    (hash md5 |M7cDVmX3r4xmab2rxYqyNg==|)
    (tag (ftp db.acme.com root)))
  (entry
    (hash md5 |kuXyqx8jYWdZ/j7Vffr+yg==|)
    (propagate)
    (tag (http http://www.internal.acme.com/accounting/)))

{KDM6YWNsKDU6ZW0cnk0NDpuYW1lKDQ6aGFzaDM6bWQlMTY6plisZirSN3C
BscfNQSbiDCKxODpzeXNh2Z1pi9vcGVyYXRvcmPkdM6dGFnKDM6ZnRwMTE
6ZGIuYWNtZS5jb200OnJvb3QpKSkoNTplbnRyeSg0Omhhc2gOzm1kNTE2Oj0
3A1Z196+MZmm9q8WKS+yKdM6dGFnKDM6ZnRwMTE6ZGIuYWNtZS5jb200OnJ
vb3QpKSkoNTplbnRyeSg0Omhhc2gOzm1kNTE2Opl8qsfI2Fw4+1X36/so
pKdK6cHJvcGFnYXR1KSgzOnRh2Zyg0Omh0dHA0MDpodHRwOi8vd3d3LmudGV
ymbFsLmFjbWUuY29tL2FjY291bnRpbmcvKSkpKQ==)
```
6.2 <sequence>

A <sequence> is a bundled sequence of objects that the verifier is to consider when deciding to grant access. We anticipate having the prover (who constructs and submits the <sequence>) provide elements in order, so that the verifier need only process the <sequence> in order while proving to itself that the prover has the claimed access right, but that is a developer decision.

The sequence can also contain instructions to the verifier, in the form of opcodes. At present the only opcode defined is "hash" -- meaning, that the previous item in the sequence (the last one read in) is to be hashed by the given algorithm and saved, indexed by that hash value. Presumably, that item (certificate body or public key, for example) is referred to by hash in some subsequent object.

At this time, we assume that <signature> does double duty, calling for the hash of the preceding item. However, it would not hurt to use an explicit <hash-op> prior to a <signature>.

If an object will be referenced by different hashes, it can be followed by multiple <hash-op>s.

Additional <op>s might be defined for some algorithms doing threshold-subject reduction (e.g., an <op> to push the current 5-tuple on a stack).
7. On-line test reply formats

An on-line test results in a digitally signed object carrying its own date range, explicitly or implicitly. That object specifies either a list of invalid certificates or that a given certificate (or list of certificates) is still valid.

This section does not give details of protocols for connecting to online servers or transmitting messages between them.

7.1 CRL and delta-CRL

If one wants to provide CRLs, and that CRL grows, then one may prefer to send only a delta CRL.

\[
\text{crl}:: \text{("crl" <version>? <hash-list> <valid-basic> ")} ;
\text{hash-list}:: \text{("canceled" <hash>*" )"} ;
\text{delta-crl}:: \text{("delta-crl" <version>? <hash-of-crl> <hash-list> <valid-basic> ")"} ;
\text{hash-of-crl}:: <hash> ;
\]

The \text{hash-of-crl} should probably have a URI pointing to the location of the full CRL.

The \text{crl} or \text{delta-crl} should be signed by the principal indicated in the (online...) field which directed the CRL to be fetched.

The CRL request can be a straight HTTP transaction, using the URI provided in the certificate, but we do not specify online protocols in this draft.

The protocol for choosing between delta and full CRL is left open. One can always provide the delta and let the caller fetch the full specifically, for example.

7.2 Revalidation

\[
\text{reval}:: \text{("reval" <version>? <subj-hash> <valid-basic> ")"} ;
\text{subj-hash}:: \text{("cert" <hash> ")"} ;
\]

This construct specifies the hash of the current certificate as \text{subj-hash} and gives a new validity period for that certificate. It should be signed by the \text{principal} indicated in the (online...) field which directed it to be fetched.
7.3 One-time revalidation

For one-time revalidation, the verifier itself must fetch the (reval) record, which will have the form:

<reval>:: "(" "reval" <version>? <subj-hash> <one-valid> ")" ;

<one-valid>:: "(" "one-time" <byte-string> ")" ;

where the byte string inside <one-valid> is one provided by the caller, expected to be unique over time and unguessable -- e.g., a large random number or random number plus sequence number. This reply should be signed by the <principal> indicated in the (online..) field which directed it to be fetched.

This result corresponds to a 0-length validity interval of "now", however the developer wishes to express that.
8. 5-Tuple Reduction

This section describes the operation of the trust evaluation machinery assumed to be part of every verifier which accepts SPKI certificates. The inputs to that trust engine are 5-tuples and any kind of certificate, not just SPKI, as well as Access Control List (ACL) entries can be translated to 5-tuples so that they can all participate in the trust computation.

A 5-tuple is an internal construct and therefore best described by a programming language data structure. A separate document will give the 5-tuple reduction code and those data structures.

Name reduction is specified in section 5.3. Therefore, in what follows we assume all issuers and subjects are principals. We also assume that all principals are public keys. It is an implementation decision whether to store these as explicit keys, hashes of keys (used as pointers) or addresses pointing to keys.

8.1 <5-tuple> BNF

How a 5-tuple is represented and stored is up to the developer. For the sake of discussion, we assume a 5-tuple is a construct of the form:

```
<5-tuple>:: <issuer5> <subject5> <deleg5> <tag-body5> <valid5> ;
<issuer5>:: <key5> | "self" ;
<subject5>:: <key5> | <obj-hash> | <keyholder> | <threshold-subj> ;
<deleg5>:: "t" | "f" ;
<key5>:: <pub-key> ;
<valid5>:: <valid-basic> | "null" | "now" ;
<tag-body5>:: <tag-body> | "null" ;
```

The extra option for issuer, "self", is provided for ACL entries. The self referred to is the verifier, holding that ACL and doing the verification of offered proofs.

The only 5-tuples that can mean anything to the verifier, after reduction is done, are those with "self" as issuer.
8.2 Top level reduction rule

\(<i1,s1,d1,a1,v1> + <i2,s2,d2,a2,v2> \) yields \(<i1,s2,d2,a,v>\) if:

\(s1 = i2\)
\(d1 = "t"\)
\(a = \) the intersection of \(a1\) and \(a2\)
\(v = \) the intersection of \(v1\) and \(v2\)

Validity intersection involves normal intersection of date ranges, if there are not-before or not-after fields in \(v1\) or \(v2\), and union of on-line tests, if those are present in \(v1\) or \(v2\). Each on-line test includes a validity period, so there is a resulting validity interval in terms of dates. This can include the string "now", as the product of a one-time on-line test result. "now" intersects with any date range to yield either "now" or "null".

The intersection of \(a1\) and \(a2\) is given below. In the most basic case,

If \(a1\) is \((\text{tag} (*))\), \(a = a2\).

If \(a2\) is \((\text{tag} (*))\), \(a = a1\).

If \(a1 == a2\), \(a = a2\).

Otherwise, \(a = "null"\) and the 5-tuple doesn’t reduce.

8.3 Intersection of tag sets

Two \(<\text{tag}>\) S-expressions intersect by the following rules. Note that in most cases, one of the two tag S-expressions will be free of *-forms. A developer is free to implement general purpose code that does set-to-set reductions, for example, but that is not likely to be necessary.

1. basic: if \(a1 == a2\), then the result is \(a1\).

2. basic: if \(a1 != a2\) and neither has a *-form, then the result is "null".

3. \((\text{tag} (*))\): if \(a1 == (\text{tag} (*))\), then the result is \(a2\).
   If \(a2 == (\text{tag} (*))\), then the result is \(a1\).

4. \((\ast \text{ set } ...\)\): if some \(<\text{tag}>\) S-expression contains a \((\ast \text{ set }\) construct, then one expands the set and does the intersection of the resulting simpler S-expressions.

5. \((\ast \text{ range } ...\)\): if some \(<\text{tag}>\) field compares a \((\ast \text{ range }\) to a
<byte-string>, one does the specified range comparison and the resulting field is the explicit one tested.

6. (* prefix ...): if some <tag> field compares a (* prefix ) to a <byte-string>, then the result is the explicit string if the test string is a prefix of it and otherwise "null".

8.4 Reduction of (subject (threshold ..))

A separate document will give full algorithms for reduction of K-of-N threshold subjects. One general procedure is to make K copies of of the 5-tuple containing the K-of-N subject and indicate which of those subjects is being handled by that copy. One then reduces that copy as if it had a single subject. One can stop the separate reductions when all K of the reduced values have the same subject. At that point, the K reduced 5-tuples become a single 5-tuple.

The actual algorithm choices for doing this reduction depend on whether one wants to reduce left-to-right or right-to-left and how much storage a verifier has.

8.7 Certificate Result Certificates

In cases where the verifier, Self, has access to a private key, once it has reduced a chain of certificate bodies down to the form:

(Self,X,D,A,V)

it can sign that generated body, using its private key, producing an SPKI certificate. That certificate will have a validity period no larger that of any certificate in the loop which formed it, but during that validity period it can be used by the prover instead of the full chain, when speaking to that particular verifier. It is good only at that verifier (or at another which trusts that verifier, Self, to delegate the authorization A). Therefore, one option by the verifier is to sign and return the result 5-tuple to the caller for this later use.

If it isn’t important for any other verifier to accept this "result certificate", it can even be signed by a symmetric key (an HMAC with secret key private to the verifier), although such keys are not defined in this standard.

The certificates which made up the loop forming this result 5-tuple could have been of any variety, including X.509v1, X.509v3, SET or
DNSSEC. They could also be PGP signed keys processed by an enriched trust engine (one capable of dealing with the PGP web of trust rules). If the verifier, Self, were to be trusted to delegate the resulting authorization, its certificate result certificate then becomes a mapping of these other forms. This may prove especially useful if a given certificate chain includes multiple forms or if the result certificate is to be used by a computationally limited device (such as a Smart-Card) which can not afford the code space to process some of the more complex certificate formats.
9. Full BNF

The following is the BNF of canonical forms and includes lengths for each explicit byte string. So, for example, "cert" is expressed as "4:cert".

9.1 Top Level Objects

The list of BNF rules that follows is sorted alphabetically, not grouped by kind of definition. The top level objects defined are:

<5-tuple>: an object defined for documentation purposes only. The actual contents of a 5-tuple are implementation dependent.

<acl>: an object for local use which might be implementation dependent. An ACL is not expected to be communicated from machine to machine.

<crl>, <delta-crl> and <reval>: objects returned from on-line tests.

<sequence>: the object carrying keys, certificates and on-line test results from prover to verifier.

9.2 Alphabetical List of BNF Rules

<5-tuple>:: <issuer5> <subject5> <deleg5> <tag-body5> <valid5> ;
<acl-entry>:: "(" "entry" <subj-obj> <deleg>? <tag> <valid>? <comment>? ")" ;
<acl>:: "(" "acl" <version>? <acl-entry>* ")" ;
<byte-string>:: <bytes> | <display-type> <bytes> ;
<bytes>:: <decimal> ":" {binary byte string of that length} ;
<cert-display>:: "(" "display" <byte-string> ")" ;
<comment>:: "(" "comment" <byte-string> ")" ;
<crl>:: "(" "crl" <version>? <hash-list> <valid-basic> ")" ;
<date>:: <byte-string> ;
<deleg5>:: "t" | "f" ;
<deleg>:: "(" "propagate" ")" ;
<delta-crl>:: "(" "delta-crl" <version>? <hash-of-crl> <hash-list> <valid-basic> ")" ;
<display-type>:: "(" <bytes> ")" ;
<fq-name>:: "(" "name" <principal> <names> ")" ;
<general-op>:: "(" "do" <byte-string> <s-part>* ")" ;
<gte>:: "g" | "ge" ;
<hash-alg-name>:: "md5" | "sha1" | <uri> ;
<hash-list>:: "(" "canceled" <hash>* ")" ;
<hash-of-crl>:: <hash> ;
<hash-of-key>:: <hash> ;
<hash-op>:: "(" "do" "hash" <hash-alg-name> ")" ;
<hash-value>:: <byte-string> ;
<hash>:: "(" "hash" <hash-alg-name> <hash-value> <uris> ")" ;
<issuer-loc>:: "(" "issuer-info" <uris> ")" ;
<issuer-name>:: "(" "issuer" "(" "name" <principal> <byte-string> ")" ")" ;
<issuer5>:: <key5> | "self" ;
<issuer>:: "(" "issuer" <principal> ")" ;
<k-val>:: <byte-string> ;
<key5>:: <pub-key> ;
<keyholder-obj>:: <principal> | <name> ;
<keyholder>:: "(" "keyholder" <keyholder-obj> ")" ;
<low-lim>:: <gte> <byte-string> ;
<lte>:: "l" | "le" ;
<n-val>:: <byte-string> ;
<name-cert>:: "(" "cert" <version>? <cert-display>? <issuer-name> <subject> <valid> <comment>? ")" ;
<name>:: <relative-name> | <fq-name> ;
<names>:: <byte-string>+ ;
<not-after>:: "(" "not-after" <date> ")" ;
<not-before>:: "(" "not-before" <date> ")" ;
<nzddigit>:: "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
<obj-hash>:: "(" "object-hash" <hash> ")" ;
<one-valid>:: "(" "one-time" <byte-string> ")" ;
<online-test>:: "(" "online" <online-type> <uris> <principal> <s-part>* ")" ;
<op>:: <hash-op> | <general-op> ;
<principal>:: <pub-key> | <hash-of-key> ;
<pub-key>:: "(" "public-key" <pub-sig-alg-id> <s-expr>* <uris> ")" ;
<pub-sig-alg-id>:: "rsa-pkcs1-md5" | "rsa-pkcs1-shal" | "rsa-pkcs1" | "dsa-sha1" | <uri> ;
<range-ordering>:: "alpha" | "numeric" | "time" | "binary" | "date" ;
<relative-name>:: "(" "name" <names> ")" ;
<reval-body>:: "(" "one-valid" | <valid-basic> ;
<reval>:: "(" "reval" <version>? <subj-hash> <reval-body> ")" ;
<s-expr>:: "(" <byte-string> <s-part>* ")" ;
<s-part>:: <byte-string> | <s-expr> ;
<seq-ent>:: <cert> | <name-cert> | <pub-key> | <signature> | <op> | <reval> | <crl> | <delta-crl> ;
<sequence>:: "(" "sequence" <seq-ent>* ")" ;
<sig-val>:: <s-part> ;
<signature>:: "(" "signature" <hash> <principal> <sig-val> ")" ;
<simple-tag>:: "(" <byte-string> <tag-expr>* ")" ;
<subj-hash>:: "(" "cert" <hash> ")" ;
<subj-obj>:: <principal> | <name> | <obj-hash> | <keyholder> | <subj-thresh> ;
<subj-thresh>:: "(" "k-of-n" <k-val> <n-val> <subj-obj>* ")" ;
<subject-loc>:: "(" "subject-info" <uris> ")" ;
<subject5>:: <key5> | <fq-name5> | <obj-hash> | <keyholder> | <subj-thresh> ;
<subject>:: "(" "subject" <subj-obj> ")" ;
<tag-body5>:: <tag-expr> | "null" ;
<tag-expr>:: <simple-tag> | <tag-set> | <tag-string> ;
<tag-prefix>:: "(" "prefix" <byte-string> ")" ;
<tag-range>:: "(" "range" <range-ordering> <low-lim>? <up-lim>? ")" ;
<tag-set>:: "(" "set" <tag-expr>* ")" ;
<tag-star>:: "(" "tag" "(*)" ")" ;
<tag-range>:: <byte-string> | <tag-range> | <tag-prefix> ;
<tag>:: <tag-star> | "(" "tag" <tag-expr> ")" ;
<up-lim>:: <lte> <byte-string> ;
<uris>:: "(" "uri" <uri>* ")" ;
<valid-basic>:: <not-before>? <not-after>? ;
<valid5>:: <valid-basic> | "null" | "now" ;
<valid>:: <valid-basic> <online-test>* ;
<version>:: "(" "version" <byte-string> ")" ;
References


[ECR] Silvio Micali, "Efficient Certificate Revocation", manuscript, MIT LCS.


[LINDEN] T. A. Linden, "Operating System Structures to Support..."


Ellison, et al.  [Page 38]
Acknowledgments

Several independent contributions, published elsewhere on the net or in print, worked in synergy with our effort. Especially important to our work were: [SDSI], [BFL] and [RFC2065]. The inspiration we received from the notion of CAPABILITY in its various forms (SDS-940, Kerberos, DEC DSSA, [SRC-070], KeyKOS [HARDY]) can not be over-rated.

Significant contributions to this effort by the members of the SPKI mailing list and especially the following persons (listed in alphabetic order) are gratefully acknowledged: Steve Bellovin, Mark Feldman, John Gilmore, Phill Hallam-Baker, Bob Juuneman, David Kemp, Angelos D. Keromytis, Paul Lambert, Jon Lasser, Jeff Parrett, Bill Sommerfeld, Simon Spero.

Authors’ Addresses

Carl M. Ellison
CyberCash, Inc.
207 Grindall Street
Baltimore MD 21230-4103 USA
Telephone: +1 410-727-4288
+1 410-727-4293 (FAX)
+1 703-620-4200 (main office, Reston, Virginia, USA)
EMail: cme@cybercash.com
cme@acm.org
Web: http://www.clark.net/pub/cme

Bill Frantz
Electric Communities
10101 De Anza Blvd.
Cupertino CA 95014
Telephone: +1 408-342-9576
Email: frantz@netcom.com

Butler Lampson
Microsoft
180 Lake View Ave
Cambridge MA 02138

Telephone: +1 617-547-9580 (voice + FAX)
EMail: blampson@microsoft.com
Expiration and File Name

This draft expires 18 September 1998.

Its file name is draft-ietf-spki-cert-structure-05.txt