Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 31, 2014

S. Leonard
Penango, Inc.

June 29, 2014

A Uniform Resource Name (URN) Namespace for Certificates
draft-seantek-certspec-03

Abstract

Digital certificates are used in many systems and protocols to identify and authenticate parties. This document describes a Uniform Resource Name (URN) namespace that identifies certificates. These URNs can be used when certificates need to be identified by value or reference.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on December 31, 2014.

Copyright Notice

1. Introduction

Digital certificates are used in many systems and protocols to identify and authenticate parties. Security considerations frequently require that the certificate must be identified with certainty, because selecting the wrong certificate will lead to validation errors (resulting in denial of service), or in improper credential selection (resulting in unwanted disclosure or substitution attacks). The goal of this namespace is to provide a uniform syntax for identifying certificates with precision in Uniform Resource Identifiers (URIs), specifically Uniform Resource Names (URNs).

Using this syntax, any protocol or system that refers to a certificate in a textual format can unambiguously identify that certificate by value or reference. Implementers that parse these URNs can resolve them into actual certificates. Examples include:

```
urn:cert:SHA-1:3ea3f070773971539b9dbf1b98c54be3a4f0f3c8
urn:cert:issuersn:cn=AcmeIssuingCompany,st=California,c=US;0134F1
urn:cert:base64:MIIBHDCBxaADAgECAgIAmTAJBgcqhkjOPQQBMBAxDjAMBgNVBAMT
                BVNtYWxsMB4XDTEzMTEwNTE5MjUzM1oXDTE2MDgwMjE5MjUzM1ow
                EDEOMAwGA1UEAxMFU21hbGwwWTATBgcqhkjOPQIBBggqhkjOPQMB
                BwNCAAS2kwRQ1thNMBMUq5d_SFdFr1uDidntNjXQrc3D_QpzYWkE
                WDsxeY8xcbl2m0TBO4TJ_2CevdoOX0OMIOaqJ_TNoxAwDjAMBgNV
                HRMBAf8EAjAAMAkGByqGSM49BAEDRwAwRAIgPyF8ok6h2NxMQ4uJ
                OcGcXYcvZ1ua0kB-rIv0omHcfNECICKwpTp3LDIwhlHTQ_DulQDD
                eYn-lnYQVc2Gm1WKAuxp
```

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2. Motivation and Purpose

Although certificates have diverse applications, there has been no uniform way to refer to a certificate in text. De-facto standards such as PEM [RFC1421] and PKIX text encoding [PT] are used to include whole certificates in textual formats, but this practice is impractical for a variety of use cases. Certificates that identify long public keys (e.g., 2048-bit RSA keys) and that contain required and recommended PKIX extensions can easily exceed many kilobytes in length.

The purpose of this document is to provide a uniform textual format for identifying individual certificates. Certificate specifications,

or "certspecs", are not designed or intended to provide a search tool or query language to match multiple certificates; the goal is to replace data elements that would otherwise have to include whole certificates in order to identify them. When a URN resolver resolves a "certspec", the resolver's output is expected to be a single certificate or nothing.

2.1. Static Identification

Identifying a specific certificate by reference or value allows diverse applications to have a common syntax. For example, applications can store certspecs as local or shared preferences, so that users can edit them without resorting to application-specific storage formats or relying on the availability of particular protocols represented by URLs (such as http:, ldap:, file:, or ni: schemes). When conveyed in protocol, a certspec can identify a specific certificate to a client or server using text-based formats such as YAML, XML, JSON, and others. The format described in this document is intended to be readily reproducible by users using common certificate processing tools, so that users can easily create, recognize, compare, and reproduce them at a glance. For example, the hash-based identifications use hexadecimal encoding so that a user can easily compose or compare an URN with a simple copy-and-paste operation. Accordingly, some tradeoffs have been made in favor of human usability.

2.2. Resolution to Context-Appropriate Schemes

When the certificate represented by a certspec needs to be resolved, an application can resort to any number of schemes. For example, when the certificate is identified by hash, the application can resolve the certspec to a Named Information (ni:) URI [RFC6920] for further processing. When the certificate is identified by issuer and serial number, the application can resolve the certspec to an LDAP service (for example, ldap:///cn=ExampleCA,o=ExampleCo,st=California,c=US).

3. One-Per-Kind

A certspec is intended to identify a single certificate unambiguously. A certificate has no more than one corresponding certspec per certspec type; however, a certificate is expected to have an array of certspecs that identify the certificate. The choice of which certspec to use in a given situation is context-specific.

4. certspec Syntax

A certspec is a URN that complies with the modern URN syntax [URNBIS], with a few accommodations for usability. Following [URNBIS], NID is "cert", and the Namespace Specific String (NSS) has the ABNF below. The query and fragment productions are relevant to certspecs; these are discussed in Section 7. Rules from [RFC5234] are also used.

```
NSS              = certspec-hash / certspec-content / certspec-el     /
                   other-certspec-type ":" other-certspec-value

hexOctet         = 2HEXDIG

certspec-hash    = "SHA-1"   ":" 20hexOctet /
                   "SHA-256" ":" 32hexOctet /
                   "SHA-384" ":" 48hexOctet /
                   "SHA-512" ":" 64hexOctet

certspec-content = "hex"     ":" 1*hexOctet /
                   "base64"  ":" base64url / base64relaxed

base64url        = 1*base64urlcharP
base64relaxed    = 1*(base64urlcharP / "+" / "/")  ; not pchar

base64urlchar  = ALPHA / DIGIT / "-" / "_" / pct-encoded ; from RFC 3986

certspec-el      = "issuersn" ":" distinguishedNameUC ";" serialNumber /
                   "ski"      ":" 1*(hexOctet)

distinguishedNameUC  = 1*pchar-no-semi            /  ; close to RFC 3986
                       distinguishedNameUCrelaxed
serialNumber         = 1*hexOctet

; semicolon omitted, since it delimits serialNumber
pchar-no-semi = unreserved / pct-encoded / "!" / "$" / "&" / "'" /
                "(" / ")" / "*" / "+" / "," / "=" / ":" / "@"

distinguishedNameUCrelaxed = 1*(pchar / WSP / UTFMB) ; not pchar
                                                     ; from RFC 4512

certspec-type        = scheme                        ; from RFC 3986
certspec-value       = 1*pchar                       ; from RFC 3986

other-certspec-type  = certspec-type
other-certspec-value = certspec-value
```

Figure 1: certspec ABNF

4.1. certspec-type and certspec-value

A certspec NSS is comprised of two parts: certspec-type and certspec-value. The certspec-type identifies the certificate specification type. The acceptable characters for spec-type are the same as those in an URI scheme name (Section 3.1 in [RFC3986]); types are compared case-insensitively. The certspec-value identifies the certificate specification value. The acceptable characters for spec-value depend on the spec-type, but are never more than pchar except for relaxed human usability reasons in a few cases discussed below. To simplify processing in several other cases, characters are significantly restricted to the point that percent-encoding is prohibited. In such cases, a generator MUST NOT generate percent-encoded values, and a parser MUST treat the production as an error.

Several certspecs use hexadecimal encodings of octets. Generally: if the hex octets are malformed (whether in the source material, such as the corresponding certificate element, or in the hex text), the certspec is invalid.

5. Standard Certificate Specifications

Standard certificate specifications are intended for interchange as durable, persistent, unique, and intuitive (to users and developers) identifiers for individual certificates--the exact criteria for URNs. This section provides four cryptographic hash-based certspecs, two content-based certspecs, and two element-based certspecs.

5.1. Cryptographic Hash-Based Specifications

A cryptographic hash or "fingerprint" of a certificate uniquely identifies that certificate. For hash-based certspecs, the hash is computed over the octets of the DER encoding of the certificate, namely, the Certificate type in Section 4.1 of [RFC5280]. The certspec-value is the hexadecimal encoding of the hash value octets. For example, a 256-bit SHA-256 hash is represented by exactly 32 hex octets, or 64 hex characters.

Lexical equivalence of two hash-based certspecs that have the same certspec-type SHALL be determined by a case-insensitive comparison of certspec-values, or by converting the hexadecimal certspec-values to octets and comparing exact equivalence of the octets. A conforming implementation MUST reject values that contain non-hex digits, such as spaces, tabs, hyphens, percent-encoded characters, or anything else.

Conforming implementations to this Internet-Draft MUST process these hash-based certspecs, unless security considerations dictate

otherwise. Acceptable reasons for refusing to process a certspec include a) the local policy prohibits use of the hash, or b) the hash has known cryptographic weaknesses, such as a preimage attacks, which weaken the cryptographic uniqueness guarantees of the hash.

5.1.1. SHA-1

The certspec-type is "SHA-1". The hash is computed using SHA-1 [SHS].

5.1.2. SHA-256

The certspec-type is "SHA-256". The hash is computed using SHA-256 [SHS].

5.1.3. SHA-384

The certspec-type is "SHA-384". The hash is computed using SHA-384 [SHS].

5.1.4. SHA-512

The certspec-type is "SHA-512". The hash is computed using SHA-512 [SHS].

5.2. Content-Based Specifications

A certificate may be identified reflexively by its constituent octets. For small-to-medium certificates, identifying the certificate by embedding it in the certspec will be computationally efficient and resistant to denial-of-service attacks (by always being available). A conforming implementation MUST implement base64 and hex specs.

The octets of a certificate are the octets of the DER encoding of the certificate, namely, the Certificate type in Section 4.1 of [RFC5280]. The DER encoding includes tag and length octets, so it always starts with 30h (the tag for SEQUENCE).

Lexical equivalence of two certspecs that are value-based SHALL be determined by decoding the certspec-value to certificate octets, and comparing the octets for strict equivalence. Accordingly, it is possible that base64 and hex certspecs are lexically equivalent.

Because users may end up copying and pasting base64 or hex-encoded certificates into certspecs, and because these certspecs will routinely exceed 72 characters, a production might contain embedded whitespace. If there are contexts where line breaks or other

whitespace must be allowed for practical reasons, the implementation should consider the URN in context as "a URN, possibly with embedded whitespace (which is ignored)".

5.2.1. base64

The certspec-type is "base64". The certspec-value is the base64url encoding of the certificate octets (Section 5 of [RFC4648]), but MAY be relaxed as follows. Unlike the data: URL [RFC2397], URN NSS productions are not supposed to have the "/" character, which is integral to standard base64. On the other hand, it is anticipated that users will want to copy-and-paste base64 encoded certificates--such as those produced by PKIX text encodings--directly into base64 certspecs. Generators of base64 certspecs SHOULD emit base64url-encoded data, where the characters '-' and '_' refer to values 62 and 63, respectively, and where the trailing equal signs '=' are absent. Alternatively, generators MAY emit base64 data with precent-encoding for the non-pchar conformant characters (specifically "/"). In any event, generators MUST NOT generate non-pchar conformant characters (specifically "/"). Parsers of base64 certspecs that are not under strict URN conformance constraints MUST also accept '+' and '/' as values 62 and 63, respectively, and MUST accept trailing '=' characters in conformance with standard base64. None of '+', '/', or '=' have reserved meanings in this certspec-type. This relaxed parsing rule is reflected in the base64relaxed production of Figure 1.

Similarly, [URNBIS] states that non-reserved characters (in this case, alphanumerics) must not be "%"-encoded, but a lenient implementation MAY decode these "%"-encoded characters anyway. This document neither recommends nor discourages such leniency, but implementors should weigh the benefits and risks as discussed further in the Security Considerations (Section 11). Overall, percent-encoding in base64 certspecs is permissible because unlike most of the other certspecs, the complete base64 encoding is not expected to be human-readable or identifiable at a glance.

5.2.2. hex

The certspec-type is "hex". The certspec-value is the hexadecimal encoding of the certificate octets. Percent-encoding is not allowed; implementations MUST NOT process percent-encoded values. The reasons are because percent-encoding would reduce the human readability of the certspec, and (marginally) increase the complexity of certspec parsers.

5.3. Element-Based Specifications

A certificate may be identified by certain data elements contained within it. The following certspecs reflect the traditional reliance of PKIX [RFC5280] and CMS [RFC5652] on a certificate's issuer distinguished name and serial number, or a certificate's subject key identifier.

If some of an element-based certspec is based on the DER encoded part of a certificate, and if the encodings are incorrect, the URN is invalid.

5.3.1. issuersn: Issuer Name and Serial Number

The certspec-type is "issuersn".

The distinguishedNameUC production encodes the certificate's issuer distinguished name (DN) field in LDAP string format, whose characters are subsequently percent-encoded to conform to URN NSS syntax. The <distinguishedName> on which distinguishedNameUC is based is defined in [RFC4514], and <SEMI> is defined in [RFC4512]. [RFC4514] no longer separates relative distinguished names (RDNs) by semicolons, as required by its predecessor, [RFC2253]. Accordingly, ";" is used to separate the issuer's DN from the subject's serial number.

Care should be taken in escaping and percent-encoding the relevant characters. In particular:

> "?" is permitted in a distinguishedName, but MUST NOT appear in a URN unless it delimits the query component (see [URNBIS]). Any "?" characters in distinguished names MUST be percent-encoded when placed in the certspec-value.
>
> "#" is used as a token at the beginning of the hexstring production for attributeValue data, but MUST NOT appear in a URN unless it delimits the fragment component (see [URNBIS]). Any "#" characters in distinguished names MUST be percent-encoded when placed in the certspec-value.
>
> "\" is the escape (ESC) character in LDAP strings (see [RFC4514]), but is not in the URI repetoire. Any "\" characters MUST be percent-encoded when placed in the certspec-value.

For reference, only the following characters are permitted in distinguished names in the issuer production of a URN:

```
ALPHA DIGIT - . _ ~ ! $ & ' ( ) * + , = : @
pct-encoded (% followed by two HEXDIG)
```

If human input is anticipated, an application MAY relax its processing as suggested in Appendix A.

The serialNumber production is the hexadecimal encoding the DER-encoded contents octets of the CertificateSerialNumber (INTEGER, i.e., not the type or length octets) as specified in Section 4.1 of [RFC5280].

A conforming implementation SHOULD implement this issuersn certspec. If the implementation implements it, the implementation MUST process serial numbers up to the same length as required by Section 4.1.2.2 of [RFC5280] (20 octets), and MUST process distinguished name strings as required by [RFC4514], including the table of minimum AttributeType name strings that MUST be recognized. Additionally, implementations MUST process attribute descriptors specified in [RFC5280] (MUST or SHOULD), and [RFC5750] (specifically: E, email, emailAddress). For reference, a complete list of required attribute descriptors is provided in Appendix B. Implementations are encouraged to recognize additional attribute descriptors where possible. A sample list of such attribute descriptors is provided in Appendix C.

Lexical equivalence of two issuersn certspecs SHALL be determined by comparing the serialNumbers for exact equivalence, and comparing the issuer distinguished names for a match.

The lexical equivalence of serialNumbers SHALL be determined by a case-insensitive comparison of them, or by converting the hexadecimal text to octets and comparing exact equivalence of the octets. A conforming implementation MUST reject values that contain non-hex digits, such as spaces, tabs, hyphens, percent-encoded characters, or anything else.

The lexical equivalence of issuer distinguished names SHALL be determined by (percent-)decoding the URNs, followed by parsing the LDAP strings. The resulting distinguished names match if they satisfy the name matching requirements of [RFC5280] and [RFC4517].

5.3.2. ski: Subject Key Identifier

The certspec-type is "ski". The certspec-value is the hexadecimal encoding of the certificate's subject key identifier, which is recorded in the certificate's Subject Key Identifier extension (Section 4.2.1.2 of [RFC5280]). The octets are the DER-encoded contents octets of the SubjectKeyIdentifier (OCTET STRING) extension value. A certificate that lacks a subject key identifier cannot and MUST NOT be identified using this spec.

Lexical equivalence of two ski certspecs SHALL be determined by a case-insensitive comparison of certspec-values, or by converting the hexadecimal certspec-values to octets and comparing exact equivalence of the octets. A conforming implementation MUST reject values that contain non-hex digits, such as spaces, tabs, hyphens, percent-encoded characters, or anything else.

A conforming implementation MAY implement this ski spec.

6. Other Certificate Specifications

The additional certificate specifications in this section are provided for applications to use as local identifiers that are useful, intuitive, or supportive of legacy systems or overriding design goals. These certspecs SHOULD NOT be used for interchange.

6.1. data (Reserved)

The certspec-type is "data". This document reserves this spec-type for future use.

An implementation may embed the contents of a data URL (data URI) into the certspec-value. Specifically:

```
                  ; from RFC 2397
certspec-value = [ mediatype ] [ ";base64" ] "," data
```

See [RFC2397]. In such a case, the mediatype SHOULD be "application/pkix-cert" since the data URL components identify a certificate; however, an implementation MAY be able to support other media types so long as a single certificate is extractable from the data production.

Data URLs containing certificates generally will not conform to URN syntax "as-is". The considerations of stuffing base64-encoded content into URNs discussed in Section 5.2.1 apply to this certspec as well, bearing in mind that data URLs only contain traditional base64 (not base64url)-encoded data, or binary percent-encoded data.

Because this certspec is content-based, an implementation can determine lexical equivalence with other content-based certspecs.

6.2. dbkey (Reserved)

The spec-type is "dbkey". This document reserves this spec-type for future use.

6.3. subject (Reserved)

The certspec-type is "subject". The certspec-value is the RFC 4514 LDAP string encoding of the certificate's subject distinguished name. Characters MAY be percent-encoded; implementations MUST process the percent-encoded characters in the certspec-value before further LDAP string processing. All the considerations of encoding the issuer field in Section 5.3.1 apply to this type.

7. Query and Fragment Productions

[URNBIS] clarifies that the query and fragment productions of [RFC3986] apply to URNs. This document provides semantics for these productions, as applied to certificates.

```
; query for certspec URN
certattrs      = query  ; from RFC 3986
                        ; *( pchar / "/" / "?" )

; fragment for certspec URN
certpart       = "v" / "sn" / "sig" / "issuer" / "notBefore " /
                 "notAfter" / "subject" / "spki" /
                 "ext" *(":" extoid *(":" extpart)) /
                 "sigval" / other-certpart

extoid         = numericoid    ; from RFC 4512
extpart        = fragment      ; from RFC 3986
other-certpart = fragment      ; from RFC 3986
```

7.1. Equivalence Unaffected

As a certspec identifies a single certificate, two certspecs are identical lexically or semantically if the NSS parts identify the same certificate. The query and fragment productions do not affect this equivalence.

7.2. Query (Attributes)

A certspec URN can have attributes (i.e., metadata) that are associated with--but not instrinsic to--the certificate or its identifiers. The syntax is intended primarily to convey certificate metadata such as attributes found in PKCS #9, PKCS #11, PKCS #12, and particular implementations of cryptographic libraries. This document does not further define certattrs; the characters of certattrs can be any valid query character from [RFC3986].

7.3. Fragment

A certspec can include a fragment that identifies a part of interest within the identified certificate. For example, a user agent may wish to draw attention to the notAfter time for an expired certificate. This document defines the following fragments ("certparts"):

Identifier	Certificate Part (ASN.1 identifier)
v	tbsCertificate.version
sn	tbsCertificate.serialNumber
sig	tbsCertificate.signature; also signatureAlgorithm
issuer	tbsCertificate.issuer
notBefore	tbsCertificate.validity.notBefore
notAfter	tbsCertificate.validity.notAfter
subject	tbsCertificate.subject
spki	tbsCertificate.subjectPublicKeyInfo
ext	tbsCertificate.extensions
ext:<OID>	tbsCertificate.extensions {Extension matching extoid == extnID}*
sigval	signatureValue

* The particular extension in the Extensions SEQUENCE is identified by OID only; there are no textual identifiers.

Table 1: certparts

The certparts defined in the table above are case-insensitive. Should additional certparts be required, a future document may specify additional certparts that match the other-certpart production.

8. Registration Template

Namespace ID:
cert

Registration Information:
Version: 1
Date: 2014-06-29

Declared registrant of the namespace:
IETF

Declaration of syntactic structures:

The structure of the Namespace Specific String is provided above.

Relevant ancillary documentation:
Certificates are defined by [RFC5280] and [X.509].

Identifier uniqueness considerations:
The certspec-type is assigned by IANA through the IETF consensus process, so this process guarantees uniqueness of these identifiers. The uniqueness of the certspec-value is guaranteed by the definition of the value for the certspec-type. For cryptographic hash-based certspecs, the cryptographic hash algorithm itself guarantees uniquess. For contents-based certspecs, the inclusion of the certificate in the URN itself guarantees uniqueness. For certspecs that identify certificates by certificate data elements, as long as certificate issuers issue certificates correctly, and the resolver's database of certificates and the resolver's implementation of certification path validation [RFC5280 sec. 6] are consistent, no cert URN will identify two different certificates.

Identifier persistence considerations:
A certificate is a permanent digital artifact, irrespective of its origin. As the URN records only information that is derivable from the certificate itself, such as one of its cryptographic hashes, the binding between the URN and the certificate is permanent.
Once the set of cert URNs identify a particular certificate, that fact will never change.

Process of identifiers assignment:
Generating a certspec (cert URN) does not require that a registration authority be contacted.

Process for identifier resolution:
This Internet Draft does not specify a resolution service for certspecs. However, resolving certificate references to actual certificates is a common practice with a wide number of offline and online implementations. See for example [RFC5280] sec. 4.2.2.1.

Rules for Lexical Equivalence:
Certspecs (cert URNs) are lexically equivalent if they both have the same certspec-type (compared case-insensitively) and the same certspec-value, and therefore impliedly point to the same certificate.

Comparison of certspec-values depends on the rules of the certspec.
Additionally, the contents-based certspecs, base64 and hex (and--if implemented--the data certspec), can be compared for lexical equivalence between each other by decoding the certspec-value to the underlying DER-encoded certificate octets, and comparing these octets for exact equivalence.
Query ("certattrs") and fragment ("certpart") components do not affect certificate identification, and therefore do not affect lexical equivalence.

Certspecs are semantically equivalent if they both resolve to the same certificate.

Conformance with URN Syntax:
The URN of this namespace conforms to URN Syntax [URNBIS] and Uniform Resource Identifier (URI): Generic Syntax [RFC3986].

Validation mechanism:
Each certspec defines the validation mechanism for its respective value. It may be appreciated that validation of the URN is a completely different process from the Certification Path Validation Algorithm [RFC5280] sec. 6, which determines whether the *certificate* is valid.

Scope:
Global.

9. Use of certspec outside URN

certspec is useful wherever a system may need to include or refer to a certificate. Some implementations may wish to refer to a certificate without enabling all of the expressive power (and security considerations) of URIs. Accordingly, this section provides a uniform method for using a certspec outside of a URN. Examples:

urn:cert:SHA-1:3ea3f070773971539b9dbf1b98c54be3a4f0f3c8
urn:cert:issuersn:cn=AcmeIssuingCompany,st=California,c=US;0134F1

To use certspec outside of a URI (URN) context, simply omit the prefix "urn:cert:". All other lexical rules apply, including percent-encoding, query (certattrs), and fragment (certparts). Care should be taken to process "?" and "#" in particular, since they delimit the attributes and parts. A conforming implementation of raw certspecs MUST permit the prefix "urn:cert:" in addition to the raw certspec. Additionally, this document guarantees that the the certspec-types "urn" and "cert" are RESERVED and will never be used.

However, implementors must take note that a raw certspec is not a valid URI, because certspec-types are not registered URI schemes and do not have the same semantics as URIs.

10. IANA Considerations

This document requests the assignment of formal URN namespace ID "cert".

[[TODO: Consider...This document requests the creation of a registry to record specs.]] New certspec types shall be ratified by the IETF consensus process. [[Some commenters have suggested the creation of a registry for certspec types. This is under consideration. One drawback is that it is desirable to limit the certspec types for interoperability and recognizability reasons--probably the only reason to include more types is for using new hashes as old hash algorithms become cryptanalyzed. The current view of the author is that no registry should be created.]]

11. Security Considerations

Digital certificates are important building blocks for authentication, integrity, authorization, and (occasionally) confidentiality services. Accordingly, identifying digital certificates incorrectly can have significant security ramifications.

When using hash-based certspecs, the cryptographic hash algorithm MUST be implemented properly and SHOULD have no known attack vectors. For this reason, algorithms that are considered "broken" as of the date of this Internet-Draft, such as MD5 [RFC6151], are precluded from being valid certspecs. The registration of a particular algorithm spec in this namespace does NOT mean that it is acceptable or safe for every usage, even though this Internet-Draft requires that a conforming implementation MUST implement certain specs.

When using content-based certspecs, the implementation MUST be prepared to process URNs of arbitrary length. As of this writing, useful certificates rarely exceed 10KB, and most implementations are concerned with keeping certificate sizes down. However, a pathological or malicious certificate could easily exceed these metrics. If an URN resolver cannot process a URN's full length, it MUST reject the certspec.

When using element-based certspecs, the implementation MUST be prepared to deal with multiple found certificates that contain the same certificate data, but are not the same certificate. In such a case, the implementation MUST segregate these certificates so that it only resolves the URN to certificates that it considers valid or

trustworthy (as discussed further below). If, despite this segregation, multiple valid or trustworthy certificates match the certspec, the certspec MUST be rejected, because a certspec is meant to identify exactly one certificate (not a family of certificates).

Certificates identified by certspecs should only be used with an analysis of their validity, such as by computing the Certification Path Validation Algorithm ([RFC5280] sec. 6) or by other means. For example, if a certificate database contains a set of certificates that it considers inherently trustworthy, then the inclusion of a certificate in that set makes it trustworthy, regardless of the results of the Certification Path Validation Algorithm. Such a database is frequently used for "Root CA" lists.

12. References

12.1. Normative References

[LDAPDESC] IANA, "LDAP Parameters: Object Identifier Descriptors", <http://www.iana.org/assignments/ldap-parameters#ldap-parameters-3>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.

[RFC2397] Masinter, L., "The "data" URL scheme", RFC 2397, August 1998.

[RFC3406] Daigle, L., van Gulik, D., Iannella, R., and P. Faltstrom, "Uniform Resource Names (URN) Namespace Definition Mechanisms", BCP 66, RFC 3406, October 2002.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4512] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Directory Information Models", RFC 4512, June 2006.

[RFC4514] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, June 2006.

[RFC4517] Legg, S., "Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules", RFC 4517, June 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5750] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling", RFC 5750, January 2010.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard", Federal Information Processing Standard (FIPS) 180-4, March 2012, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

[URNBIS] Saint-Andre, P., "Uniform Resource Name (URN) Syntax", draft-ietf-urnbis-rfc2141bis-urn-07 (work in progress), January 2014.

12.2. Informative References

[PT] Josefsson, S. and S. Leonard, "Text Encodings of PKIX and CMS Structures", draft-josefsson-pkix-textual-02 (work in progress), April 2014.

[RFC1421] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC 1421, February 1993.

[RFC2253] Wahl, M., Kille, S., and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, March 2011.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, April 2013.

Appendix A. Relaxed Processing for Issuer Distinguished Name

If human input is anticipated, an application may relax its processing of the issuer distinguished name in the issuersn spec. The following techniques will not produce a strictly conforming certspec URN, but may prove useful in mapping borderline inputs to valid URNs (and therefore, to specific certificates). Most of these techniques are reflected in the distinguishedNameUCrelaxed production.

A real-world LDAP string will likely contain spaces, such as between words. Parsers SHOULD accept spaces when parsing this certspec; generators MAY emit spaces when strict conformance to URN syntax is less important than human readability (for example, when the URN is rendered for display, or in cases where the URN is expected to be handled by humans).

Distinguished name attribute values may include Unicode characters outside of the US-ASCII range (0x00-0x7F), as well as characters that need to be escaped with [RFC4514] rules. The interaction between URNs, LDAP strings, and human usability allows for multiple representations of these characters, two of which are strictly conformant and one of which should be anticipated for human input.

At the LDAP string level, a non-ASCII character can be a UTF-8 sequence, or can be escaped with "\" followed by two hex digits for each UTF-8 octet in the sequence. At the URN level, a UTF-8 sequence must be converted to "%" followed by two hex digits for each UTF-8 octet in the sequence; if the characters are already escaped, "\" must be converted to %5C.

Example Attribute:

```
        Name: sn
       Value: E. Mu1oz$el<Toro?
          <1> is actually U+00F1 Latin Small Letter N With Tilde,
              UTF-8 encoded as octets 0xC3 0xB1.
           $  is sub-delim; it may appear in a URN assigned-name.
           <  is required to be escaped per LDAP string rules.
           ?  is gen-delim; it may not appear in a URN assigned-name.
```

Conformant LDAP Strings:

```
    sn=E. Mu1oz$el\<Toro?
    sn=E. Mu\C3\B1oz$el\<Toro?
```

Conformant URN Productions:

```
    urn:cert:issuersn:sn=E.%20Mu%C3%B1oz$el%5C%3CToro%3F;22
    urn:cert:issuersn:sn=E.%20Mu%5CC3%5CB1oz$el%5C%3CToro%3F;22
```

Humans will likely supply (and UIs will likely display) characters without the requisite escaping. Therefore, a parser that accepts human input SHOULD be prepared to accept raw UTF-8 characters and reserved characters that are not percent-encoded per URN rules. However, such a parser SHOULD strictly reject sequences that do not conform to LDAP string [RFC4514] rules:

```
    urn:cert:issuersn:sn=E. Mu1oz$el\%3CToro%3F;22
```

In the example above, " " is not percent-encoded, 1 (n with tilde) is neither LDAP-escaped nor percent-encoded, and \ is not percent-encoded. Contextually, however, the intent is obvious. In contrast, the escape character \ (whether or not percent-encoded) SHOULD precede < because without it, the string does not conform to [RFC4514]. The ? character SHOULD remain percent-encoded as %3F because otherwise the suffix ;22 would be interpreted as the query component.

URNs supplied by human input may include ";" as a delimiter between attributes, or if escaped, within attribute values. A strictly conformant certspec parser will reject such inputs. However, a parser specifically designed to process issuer distinguished names MAY distinguish these semicolons from the serial number separator by searching backwards in the string, skipping any query or fragment components. The last semicolon would be treated as the separator, while other semicolons would be treated as parts of the issuer LDAP string.

Appendix B. Mandatory Attribute Descriptors for issuersn certspec

As per [RFC4514], attribute descriptors case-insensitive. A conformant implementation MUST recognize the attributes in the table below, both by the OIDs and by the names recorded in the LDAP Parameters: Object Identifier Descriptors registry [LDAPDESC].

OID	Names	RFC
2.5.4.3	cn (CN) commonName	4514

2.5.4.7	l (L) localityName	4514
2.5.4.8	st (ST) (S)* stateOrProvinceName	4514
2.5.4.10	o (O) organizationName	4514
2.5.4.11	ou (OU) organizationalUnitName	4514
2.5.4.6	c (C) countryName	4514
2.5.4.9	street (STREET) streetAddress	4514
0.9.2342.19200300.100.1.25	dc (DC) domainComponent	4514
0.9.2342.19200300.100.1.1	uid (UID) userId	4514
2.5.4.5	serialNumber (SERIALNUMBER)	5280
2.5.4.46	dnQualifier (DNQUALIFIER)	5280
2.5.4.4	sn (SN) surname	5280
2.5.4.42	gn (GN)** givenName	5280
2.5.4.12	(T)* title	5280
2.5.4.43	(I)* initials	5280
2.5.4.44	(GENQUALIFIER)* generationQualifier (GENERATIONQUALIFIER)	5280
2.5.4.65	(PNYM)* pseudonym (PSEUDONYM)	5280
1.2.840.113549.1.9.1	(E)* emailAddress email	5750

Names in parentheses are variations that are not assigned as such in [LDAPDESC]. Implementations MAY parse these names, but SHOULD NOT generate them.

Names in ALL-CAPS may be emitted by some certificate-processing applications; these names are compatible with lowercase or mixed-case variations due to case-insensitivity.

* Name may appear in some implementations, but is not in [LDAPDESC].

** Name commonly appears in implementations, but is RESERVED in [LDAPDESC].

Table 2: Attribute Descriptors

Appendix C. Recommended Attribute Descriptors for issuersn certspec

As per [RFC4514], attribute descriptors case-insensitive. [[TODO: complete.]]

Author's Address

Sean Leonard
Penango, Inc.
5900 Wilshire Boulevard
21st Floor
Los Angeles, CA 90036
USA

Email: dev+ietf@seantek.com
URI: http://www.penango.com/