Network Working Group S. Leonard
Internet-Draft Penango, Inc.
Intended status: Standards Track June 8, 2016
Expires: December 10, 2016

String Specification for Certificates
draft-seantek-certspec-06

Abstract

Digital certificates are used in many systems and protocols to identify and authenticate parties. This document describes a string format that identifies certificates, along with optional attributes. This string format has been engineered to work without re-encoding in a variety of protocol slots.

Table of Contents

1. Introduction

Digital certificates [RFC5280] 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 document is to provide a uniform syntax for identifying certificates with precision without re-encoding in a variety of protocol slots.

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. Implementations that parse these strings can resolve them into actual certificates. Examples include:

```
SHA-1:3ea3f070773971539b9dbf1b98c54be3a4f0f3c8
ISSUERSN:cn=AcmeIssuingCompany,st=California,c=US;0134F1
BASE64:MIIBHDCBxaADAgECAgIAmTAJBgcqhkjOPQQBMBAxDjAMBgNVBAMT
       BVNtYWxsMB4XDTEzMTEwNTE5MjUzM1oXDTE2MDgwMjE5MjUzM1ow
       EDEOMAwGA1UEAxMFU21hbGwwWTATBgcqhkjOPQIBBggqhkjOPQMB
       BwNCAAS2kwRQ1thNMBMUq5d/SFdFr1uDidntNjXQrc3D/QpzYWkE
       WDsxeY8xcbl2m0TBO4TJ/2CevdoOX0OMIOaqJ/TNoxAwDjAMBgNV
       HRMBAf8EAjAAMAkGByqGSM49BAEDRwAwRAIgPyF8ok6h2NxMQ4uJ
       OcGcXYcvZ1ua0kB+rIv0omHcfNECICKwpTp3LDIwhlHTQ/DulQDD
       eYn+lnYQVc2Gm1WKAuxp
/etc/myserver.cer|friendlyName=fluffy the Tomcat
URI:https://certificates.example.com/acme/BAADF00D.cer
```

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.

1.2. Definitions

The term "certificate" means either a certificate [RFC5280] or an attribute certificate [RFC5755]. When a certificate [RFC5280] alone is to be distinguished, this specification may use the term "public key certificate".

The term "whitespace" means HT, VT, FF, LF, CR, and SP, when referring to the ASCII range. An implementation SHOULD also consider whitespace beyond the ASCII range, if the implementation supports it, e.g., the characters that have the White_Space character property in [UNICODE].)

The term "content" means a fixed sequence of octets (i.e., data) with an Internet media type and optional parameters.

2. Motivation and Purpose

Although certificates [RFC5280] 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 [RFC7468] 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, with human usability as a design goal. 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, or that employ proprietary reference schemes. For example, certspecs fit easily into XML/SGML data, YAML, JSON, and config files and databases (e.g., .properties, .ini, and Windows Registry) with minimal required escaping.

To be usable by humans, certspecs are supposed to be amenable to copy-and-paste operations. The structure of a certspec is also supposed to be plainly visible, do that someone glancing at a certspec can ascertain the data types that it comprises.

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 URIs (such as http:, ldap: [RFC4516], file: [RFC1738], 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.

2.2. Relationship with Other Specifications

Previous versions of this draft attempted to define certspecs as a URN namespace, and then as a family of URI schemes. Certspecs were found to be incompatible with these approaches for several engineering reasons.

The definition of URN [I-D.ietf-urnbis-rfc2141bis-urn] changed during the course of this draft's development, and as of this publication, remains unclear. Overall, URNs are meant to be assigned durably and persistently by namespace authorities. An algorithm that identifies a datum with high but not absolute precision does not satisfy this requirement, because the pigeonhole principle shows that there will always be collisions for unbounded data sets, even though finding particular collisions may be computationally infeasible. (The other specification types are even less precise.)

URI schemes were also pursued and rejected. URIs have dereferencing semantics, which could have significant security implications. When a URI is dereferenced, an access mechanism is determined and an action is performed on the URI's resource (see Section 1.2.2 of [RFC3986]). When the URI scheme identifies an information retrieval protocol, such as HTTP, the access mechanisms usually involve performing actions on the resource (most commonly, retrieving the resource). However, a wide variety of standardized and proprietary URI schemes do not correspond to information retrieval protocols; instead, these URI schemes serve to launch applications associated with the scheme names. The practical effect of dereferencing a mailto: URI [RFC6068], for example, is to launch a preferred e-mail client from a user agent (e.g., a web browser), so that the user can easily send e-mail to a recipient with certain fields pre-populated by the contents of a URI. The practical effect of the ymsgr: URI [PROVURI] is much more application and vendor-specific: dereferencing such a URI is supposed to launch Yahoo! Messenger to send an instant message to a designated Yahoo! screen name. The launching semantics of URIs are exploited in modern consumer desktop and mobile operating systems as a convenient methodology to launch an application from another application, as well as to communicate application-specific data between applications that otherwise have no privileged relationship.

The arbitrary, misdirected, or outright malicious launching of applications to handle certificates has grave security implications. These risks are mitigated by avoiding URI schemes for certspecs.

URI schemes such as ni: [RFC6920] and data: identify resources in a similar way as (for example) the hash and data-based spec types, although they were not as usable for copy-and-paste operations.

Resistance was encountered when new URI scheme names were proposed that did similar things as ni: and data:, but with better usability for this use case.

Nevertheless, this draft allows for specifying a certificate by URI, recognizing that an implementation might categorically decline to retrieve URI certspecs on security or other grounds.

To distinguish this syntax from URI syntax, this Internet-Draft capitalizes the "introducer characters" of the various certspec types and does not require that they be delimited with a colon, even though these productions (mostly) are case-insensitive and (mostly) end with a colon. OpenSSL's x509v3_config format inspired this aspect of the syntax.

3. certstring Syntax

A certificate string ("certstring") is a string with a single certspec (see Section 4) or multiple certspecs (a "multispec", see Section 7), followed by an optional set of attributes ("certattrs", see Section 8). While strings in this document can be in any character encoding, the delimiter characters in this document are drawn from ASCII; applications MUST support Unicode. The string has the ABNF [RFC5234] below (TODO: case-sensitive ABNF, import core rules).

```
certstring = (certspec / multispec) [ "|" certattrs ]
```

Figure 1: certstring ABNF

4. certspec Syntax

A certspec is a string that is intended to identify a single certificate. A certspec has introducer characters followed by value characters; these introducer characters MAY be part of the "value" of the identifier. The string has the ABNF [RFC5234] below (TODO: case-sensitive ABNF, import core rules).

```
certspec         = certspec-hash / certspec-content / certspec-el /
                   certspec-path

hexOctet         = 2HEXDIG

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

```
certspec-other-hash = certspec-hash-type ":" certspec-hash-value
; Proposal: Hash Function Textual Name registry hereby limited
; to RFC 3986 scheme characters
certspec-hash-type = scheme
; implication is that it must be at least 128 bits
certspec-hash-value = 16*hexOctet

certspec-content = ("HEX" / "BASE16") ":" 1*hexOctet /
                   "BASE64"  ":" base64string

base64char       = ALPHA / DIGIT / "+" / "/"
base64string     = 1*(4base64char)
                   [ 3base64char "=" / 2base64char "==" ]

; distinguishedName from [RFC4514]
certspec-el   = "ISSUERSN" ":" distinguishedName ";" serialNumber /
                "SKI"      ":" 1*(hexOctet)

serialNumber     = 1*hexOctet

certspec-path    = certspec-uri / certspec-filepath

; from RFC3986; RFC 6570
certspec-uri     = "URI:" URI-reference / URI-Template

; see POSIX, etc.
certspec-filepath = ("/" / "\" / [A-Z] ":" /
                    ("." / "..") ("/" / "\") / "~" / "%" / "$")
                    *filepathchar

; BEYONDASCII is from draft-seantek-more-core-rules
filepathchar      = %x01-29 / %x2B-3B / "=" / %x40-5B /
                    %x5D-7B / %x7D-7F / quoted-fpc / BEYONDASCII

quoted-fpc       = "\" ("*" / "<" / ">" / "?" / "\" / "|")

; TODO: validate Windows file path characters

; TODO: certattrs
```

Figure 2: certspec ABNF

4.1. certspec Type and Value

Semantically, a certspec is comprised of its type and value. The value is always provided, but the type is either explicitly declared, or is inferred from the initial (introducer) characters in the type. When types are explicitly provided, they are compared case-

insensitively. The certspec-value identifies the certificate specification value.

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 intuitive (to users and developers) identifiers for individual certificates. This section provides four cryptographic hash-based certspecs, two content-based certspecs, three element-based certspecs, and two path-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] and the AttributeCertificate type in Section 4.1 of [RFC5755]. 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. The hexadecimal encoding is not case sensitive.

A conforming generator SHALL emit only hexadecimal encoded data, i.e., the characters A-F (case-insensitive) and 0-9.

A conforming parser SHALL accept value productions that contain the following non-hex digits: whitespace, hyphen, and colon. A conforming parser MAY accept values that contain other characters.

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 introducer production is "SHA-1:". The hash is computed using SHA-1 [SHS].

5.1.2. SHA-256

The introducer production is "SHA-256:". The hash is computed using SHA-256 [SHS].

5.1.3. SHA-384

The introducer production is "SHA-384:". The hash is computed using SHA-384 [SHS].

5.1.4. SHA-512

The introducer production is "SHA-512:". The hash is computed using SHA-512 [SHS].

5.2. Content-Based Specifications

Content-based certspecs identify certificates by their 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] and the AttributeCertificate type in Section 4.1 of [RFC5755]. The DER encoding includes tag and length octets, so it always starts with 30h (the tag for SEQUENCE).

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. A conforming generator SHALL emit no whitespace, or SHALL emit a hanging indent, between semantically significant characters.

5.2.1. BASE64

The introducer production is "BASE64:". The value production is the BASE64 encoding of the certificate octets (Section 4 of [RFC4648]).

[[NB: base64url syntax was explicitly considered and rejected for this draft. This is because certspecs are no longer URIs.]]

5.2.2. HEX and BASE16

The introducer production is "HEX:" or "BASE16:". Generators MUST generate "HEX:"; parsers MUST accept "HEX:" and "BASE16:". The value production is the hexadecimal encoding of the certificate octets.

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, a certificate's subject distinguished name and expiration, or a certificate's subject key identifier.

Note that distinguished names can contain "|" in attribute value strings, but this production is unambiguous with the certattr delimiter because distinguished names are always terminated by ";".

5.3.1. ISSUERSN: Issuer Name and Serial Number

The introducer production is "ISSUERSN:".

5.3.1.1. Issuer

The distinguishedName production encodes the certificate's issuer distinguished name (DN) field in LDAP string format [RFC4514]. [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.

5.3.1.2. Serial Number

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 4.1.2.2 of [RFC5280].

5.3.1.2.1. Conformance

A conforming implementation SHALL implement the ISSUERSN certspec. An 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. Conforming implementations MUST be able to parse all distinguished name attribute types that are encoded in OID dotted decimal form, as well as all distinguished name attribute values that are encoded in "#" hexadecimal form.

5.3.1.3. SUBJECTEXP: Subject and Expiration

The introducer production is "SUBJECTEXP:". The value production is the [RFC4514] encoding of the subject distinguished name, followed by a semicolon, followed by the certificate expiration expressed in standard form [[TODO: ASN.1 text form, or RFC3339 form? Proposal is to allow both. ASN.1 text form: GeneralizedTime with four-digit years (UTCTime/four-digit years SHALL NOT be used), in accordance with Section 4.1.2.5.2 of [RFC5280]. RFC3339 form: The date-time production from [RFC3339]. (The production requires the full four-digit year, but allows for a time zone offset. Time zone offsets MUST be supported on usability grounds.)]] The certificate's expiration is the notAfter value of the certificate validity period (Section 4.1.2.5 of [RFC5280]).

A conforming implementation SHALL parse and generate distinguished name productions with the same adherence as stated above in Section 5.3.1.2.1.

5.3.1.4. ski: Subject Key Identifier

The introducer production is "SKI:". The value production 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. For a certificate that lacks a subject key identifier, an underlying implementation MAY operatively associate a subject key identifier with the certificate.

A conforming generator SHALL emit only hexadecimal encoded data, i.e., the characters A-F (case-insensitive) and 0-9.

A conforming parser SHALL accept value productions that contain the following non-hex digits: whitespace (HT, VT, SP, FF, CR, LF), hyphen, and colon. A conforming parser MAY accept values that contain other characters.

5.4. Path-Based Specifications

A certificate may be identified by a path to file or content data. A conforming parser MUST recognize file path and URI specs, although conforming implementations merely MAY process them.

5.4.1. File Path

File paths are identified by their introducer productions / \ [A-Z]: ./ ../ .\ ..\ ~ % and $. The characters that follow MUST be valid path characters for the system on which the files are being accessed. Since the starting character sequences for file paths are fixed and determinable, prefixing the file path with a type identifier is (thought to be) unnecessary.

A relative file path begins with "." or "..", and is relative to a "current directory". Determining an appropriate "current directory" is outside the scope of this specification.

When the file is read, implementations MUST accept the following, regardless of filename:

1. Textual data, which is analyzed as if it were text/plain content (below)

2. Raw octet-oriented data, which is analyzed as if it were application/octet-stream content (below)

File paths may have unexpanded environment variables, such as %USERNAME% or ${LOGNAME}; implementations MUST parse these environment variable syntaxes, but merely MAY perform environment variable substitution as environment, capability, and security concerns dictate.

Note that Unix-oriented file paths can contain "|" in the production "\|", but this production is unambiguous with the certattr delimiter.

5.4.2. URI

The introducer production is "URI:". The value is a [RFC3986] conforming URI-reference or [RFC6570] conforming URI-Template.

In the context of URIs, a relative reference conforms to the relative-ref production of [RFC3986] and the usage described in Section 4.2 of [RFC3986], it is relative to a "base URI". Determining an appropriate "base URI" is outside the scope of this specification.

When the URI is dereferenced, implementations MUST accept the following, regardless of the path or query productions:

1. Content with media type application/pkix-cert and application/pkix-attr-cert

2. Content with media type application/pkcs7-mime and application/cms, when the content represents a SignedData containing certificates (regardless of the smime-type or encapsulatingContent parameters, and regardless of whether or not the SignedData is in a degenerate, certs-only format)

3. Content with media type text/plain, which is analyzed according to [RFC7468] for "CERTIFICATE" and "ATTRIBUTE CERTIFICATE" textual encodings

4. Content with media type application/octet-stream, which is analyzed for textual or [X.690] data

5. Raw textual data, which is analyzed as if it were text/plain

6. Raw octet-oriented data, which is analyzed as if it were application/octet-stream

The URI certspec can include a fragment identifier. Implementations MUST parse fragment identifiers, but merely MAY perform "secondary resource" isolation and processing as environment, capability, and security concerns dictate.

The URI certspec can be a URI Template [RFC6570]. Implementations MUST parse URI templates, but merely MAY expand them in accordance with [RFC6570] as environment, capability, and security concerns dictate.

Note that URI templates can contain "|" in the production "{|".."}", but this production is unambiguous with the certattr delimiter.

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. DBKEY (Reserved)

The introducer production is "DBKEY:". The DBKEY certspec is meant for an opaque string that serves as the unique key to a certificate in an implementation's certificate database. This document reserves this introducer sequence for future use.

6.2. SELECT (Reserved)

The introducer production is "SELECT" (without a colon). The SELECT certspec is meant for a valid SQL statement (suitably escaped) that retrieves a row representing a certificate. This document reserves this introducer sequence for future use.

7. Multiple certspecs (multispec)

A multispec is a string that contains multiple certspecs, each of which is intended to identify the exact same certificate. If multiple certificates match a single spec, a single certificate can be returned by the access operation, so long as the intersection of certificates identified by all of the certspecs in the multispec is one. The purpose of multispec is to provide multiple access and verification methods. For example, a hash algorithm may have known weaknesses, but may be the most efficient way to identify a certificate (e.g., because it is the index method). Providing additional certspecs (i.e., strong hash algorithms) would increase the certainty that the correct certificate is accessed.

As the certspecs above make use of almost all other characters in the ASCII range, < and > have been chosen to delimit certspecs between each other. (Whitespace can also appear between each < and > delimited certspec.) The ABNF of multispec is:

```
multispec = 1*("<" certspec ">")
```

Figure 3: multispec ABNF

8. Certificate Attributes (certattrs)

A certificate can have additional attributes (i.e., metadata) operatively associated with--but not intrinsic to--it. For example, the additional attributes may represent preferences. The syntax is intended primarily to convey certificate metadata such as attributes found in PKCS #9 [RFC2985], PKCS #11 [PKCS11], PKCS #12 [RFC7292], and particular implementations of cryptographic libraries.

Certattrs are delimited from a certspec or multispec production with "|". Each certattr SHALL have a corresponding ASN.1 definition. The

textual syntax of certattrs is very similar to (in fact, a superset of) [RFC4514]: the certattrs production represents the PKCS Attributes family of types, which are repeatedly defined in those standards, and standards that derive from them, as SET SIZE (1..MAX) OF Attribute. E.g., CMS (from PKCS #7) [RFC5652], private keys (from PKCS #8) [RFC5958], and PKCS #12 [RFC7292]. Attributes are semantically unordered. Multiple attributes are separated with ",".

Each attribute has a single attrType (canonically defined as OBJECT IDENTIFIER in [RFC5652]), and a SET OF attrValues. The attrType is encoded as the string representation of AttributeType (that is, either a registered short name (descriptor) [RFC4520], or the dotted-decimal encoding, <numericoid> of the OBJECT IDENTIFIER [RFC4512]).

When an attribute has at least one value, the attrType is followed by "=" and the encoding of the attrValues (empty strings are possible). Multiple attrValues are separated by "+". When the attribute has no values, the attrType MUST NOT be followed by "=".

An attrValue can have one of several encodings:

hex: The attrValue can always be represented by "#" followed by the hexadecimal encoding of each of the octets of the BER encoding of the attrValue, following paragraph 1 of Section 2.4 of [RFC4514]. Implementations MUST support this encoding.

string: If the attrValue has a LDAP-specific string encoding, that encoding can be used as the string representation of the value, with characters suitably escaped according to paragraph 2 and onward of Section 2.4 of [RFC4514]. Implementations SHOULD support this encoding for attributes of interest to it.

XER: The attrValue can be represented by its BASIC-XER encoding [X.693] (Clause 8). When in BASIC-XER encoding, the string MUST be a complete XML fragment comprising one element, i.e., there SHALL NOT be an XML prolog. XER encoding is self-delimiting because it has balanced elements; this string always begins with "<" and ends with ">". Processing is simplified compared to arbitrary XML in that XML processing instructions, XML comments, and CDATA sections are prohibited. Implementations MAY support this encoding.

ASN.1 value: The attrValue can be represented by its ASN.1 value notation [X.680], enclosed in quotation marks <"> on both ends. [[NB: per RFC 4514, a leading space might also be unambiguous.]] ASN.1 value notation requires a bit of finesse in that <"> can appear inside to delimit "cstring" lexical items (see Clause 12.14 and Clause 41 of [X.680]). A "cstring" starts and ends with <">, and can

represent <"> internally with a pair of consecutive <">. Therefore, <"> is balanced because it always occurs in multiples of two. If the value is just a cstring, then the representation will have exactly two <"> at the beginning, and two <"> at the end, with evenly-balanced <"> pairs inside. Other values that are not composites (enclosed with "{" and "}") do not have <"> occur within them. Otherwise, the representation must have at least one "{" "}" balanced pair at either end, hemming in <"> occurrences to within the balanced pairs of "{" and "}". Implementations MAY support this encoding.

Of the attrValue encodings listed above, only "hex" can reliably transfer the underlying BER representation without an implementation maintaining specific knowledge of every attribute. Therefore, "hex" is RECOMMENDED for open interchange of certattrs.

8.1. ABNF

The collective ABNF of certattrs is:

```
certattrs = certattr ["," certattrs]
certattr  = certattrType ["=" certattrValues]
; descr, numericoid from RFC 4512
certattrType = descr / numericoid
certattrValues = certattrValue ["+" certAttrValues]
; string, hexstring from RFC 4514
certattrValue = hexstring / string /
                basic-xer-string / asn1-value-string
; TODO: complete basic-xer-string, asn1-value-string
basic-xer-string = "<" <balanced tags> ">"
; TODO: may also distinguish with leading space " "--think about it
asn1-value-string = %x22 <balanced quotation marks> %x22
```

Figure 4: certattrs ABNF

8.2. Mandatory Attribute Support

[[NB: attributes related to certificate objects are in the domain of CMS attributes, NOT distinguished name attributes. Therefore, referring to the LDAP Object Identifier Descriptors subregistry may actually be inappropriate, since it's pretty much filled with attributes that one would encounter for distinguished names. The "CMS attributes" encompass things like friendlyName and smimeCapabilities from PKCS #9; they are a disjoint set from distinguished name attributes. "CMS attributes" also encompass things like signingTime and messageDigest; these attributes are not interesting with respect to certificate objects.]]

A conforming implementation that supports certattrs SHALL process the following attributes, including recognizing the following short names (descriptors) and associated LDAP-specific string encodings.

friendlyName (1.2.840.113549.1.9.20) from PKCS #9 [RFC2985]

localKeyId (1.2.840.113549.1.9.21) from PKCS #9 [RFC2985]

signingDescription (1.2.840.113549.1.9.13) from PKCS #9 [RFC2985]

smimeCapabilities (1.2.840.113549.1.9.15) from PKCS #9 [RFC2985]
[[NB: smimeCapabilities does not have a SYNTAX with an LDAP-specific encoding. ASN.1 value notation is probably the most readable alternative, but support for ASN.1 value notation remains OPTIONAL.]]

8.3. Canonicalization

The certattrs production is a textual encoding of the ASN.1 SET SIZE (1..MAX) OF Attribute. The textual format in this section is not intended to be used as any kind of canonical form. The canonical form is the DER encoding of the corresponding SET SIZE (1..MAX) OF Attribute.

9. Whitespace

This specification is intended for textual data that may be visible to or edited by humans. Whitespace is a key factor in usability, so this specification permits whitespace in certain productions.

The certspec, multispec, certattrs, and certstring productions are ideally emitted as one (long) line. The overall intent is that a bare line break (without leading or trailing horizontal space) is supposed to delimit these productions from each other.

If it is desirable to break one of these productions across multiple lines, a hanging indent SHALL be used at syntactically appropriate places. A hanging indent means a newline production (LF, CRLF, or other characters appropriate to the character set, e.g., [[UNICODE]]) followed by one or more horizontal space characters. The preferred horizontal space production is a single SP character.

Generally, where whitespace is permitted, the whitespace either has no semantic meaning and therefore can be collapsed to a zero-length substring, i.e., skipped, or can be folded into a single whitespace character, i.e., a single SP.

Productions that represent the hexadecimal (or base64) encodings of octets MAY have arbitrary whitespace interspersed between the

hexadecimal (or base64) characters. The whitespace has no semantic meaning, and can be collapsed. Certspec and certattrs parsers that parse "#" delimited attribute values in distinguished names and certificate attributes MAY accept and collapse whitespace; however, such whitespace is not permitted by [RFC4514]. Note that the attribute value MUST begin with "#"; there MUST NOT be leading whitespace.

A parser MAY accept whitespace preceding the certattrType production in certattrs.

A parser MAY accept whitespace between each angle-bracket-delimited certspec in the multispec production.

A parser MAY accept whitespace preceding the attributeType production in distinguishedName.

Generally, whitespace characters in values are otherwise considered to be semantically meaningful. A generator SHOULD encode such characters (e.g., with hexpair [RFC4514]) to avoid ambiguity or corruption.

10. Guidelines for Extending certspec

The certspec definition presented in this document is intended to be fairly comprehensive. Nevertheless, there are several points of extension for implementors who may want to identify a certificate with more than what is presented in this document.

Firstly, certspec is naturally extended by supporting additional hash algorithms. The hash introducer characters are tied to the Hash Function Textual Names Registry; adding a new hash algorithm to that registry is necessary for certificates to get identified with that hash algorithm under this specification. However, for security reasons, the introducers "MD2" and "MD5" SHALL NOT be generated or parsed.

Secondly, certspec allows for the full range of "local" identifiers (i.e., file paths, which may not actually be local) and "network" identifiers (i.e., URIs, which may not actually need the network). A certspec implementation that can make use of these facilities can naturally be extended by extending the path (e.g., with pipes and mount points) or the URI topology (e.g., with novel URI schemes).

Implementations MAY recognize other types of certspecs. However, new types intended for open interchange require an update to this document.

A new certspec SHALL satisfy the following criteria:

1. The type is identified by a keyword, followed by ":", or, the type is identified by very short sequences of characters that unambiguously signal the type of the certspec value (as file paths currently do). The specification MUST state whether the introducer characters are case-sensitive.

2. The characters "<", ">", and "|" need to be distinguishable from their uses in multispec and certattrs (certstring) using a context-free grammar, e.g., ABNF.

3. [[TODO: further elaborate, or remove.]] If internal whitespace (including line-breaking) is permitted, the internal whitespace is consistent with this specification.

11. Use of certspec in Systems

certspec is useful wherever a system may need to include or refer to a certificate. Some systems may wish to refer to a certificate without enabling all of the expressive power (and security considerations) of all strings in this specification. Accordingly, those systems and specifications SHOULD develop profiles of this specification.

This document guarantees that the introducer characters "URN:" and "CERT:" are RESERVED and will never be used. Implementors MUST take note that a raw certspec is not a valid URI: certspec-types are not registered URI schemes, have a broader character repertoire than permitted by [RFC3986], and do not have the same semantics as URIs.

12. IANA Considerations

This document implies no IANA considerations.

13. 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 strings 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.

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 the implementation only continues with 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 (not in a multispec) 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 (Section 6 of [RFC5280]) 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.

14. References

14.1. 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.

[RFC2585] Housley, R. and P. Hoffman, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", RFC 2585, May 1999.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <http://www.rfc-editor.org/info/rfc3339>.

[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.

[RFC4520] Zeilenga, K., "Internet Assigned Numbers Authority (IANA) Considerations for the Lightweight Directory Access Protocol (LDAP)", BCP 64, RFC 4520, DOI 10.17487/RFC4520, June 2006, <http://www.rfc-editor.org/info/rfc4520>.

[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.

[RFC5755] Farrell, S., Housley, R., and S. Turner, "An Internet Attribute Certificate Profile for Authorization", RFC 5755, January 2010.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <http://www.rfc-editor.org/info/rfc6570>.

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <http://www.rfc-editor.org/info/rfc7468>.

[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>.

14.2. Informative References

[I-D.ietf-urnbis-rfc2141bis-urn] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", draft-ietf-urnbis-rfc2141bis-urn-16 (work in progress), April 2016.

[PKCS11] RSA Laboratories, "PKCS #11 v2.30: Cryptographic Token Interface Standard", PKCS 11, April 2009.

[PROVURI] IANA, "Uniform Resource Identifier (URI) Schemes: Provisional URI Schemes", <http://www.iana.org/assignments/uri-schemes/uri-schemes.xml#uri-schemes-2>.

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

[RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

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

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, November 2000.

[RFC4516] Smith, M. and T. Howes, "Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator", RFC 4516, June 2006.

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

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, August 2010.

[RFC6068] Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", RFC 6068, October 2010.

[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.

[RFC7292] Moriarty, K., Nystrom, M., Parkinson, S., Rusch, A., and M. Scott, "PKCS #12: Personal Information Exchange Syntax v1.1", RFC 7292, July 2014.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 8.0.0", ISBN 978-1-936213-10-8, August 2015.

Mountain View, CA: The Unicode Consortium.

[X.680] International Telecommunications Union, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, August 2015, <https://itu.int/ITU-T/X.680>.

[X.690] International Telecommunications Union, "ASN.1 encoding rules: Specification of basic encoding Rules (BER), Canonical encoding rules (CER) and Distinguished encoding rules (DER)", ITU-T Recommendation X.690, August 2015, <https://itu.int/ITU-T/X.690>.

[X.693] International Telecommunications Union, "ASN.1 encoding rules: XML Encoding Rules (XER)", ITU-T Recommendation X.693, August 2015, <https://itu.int/ITU-T/X.693>.

Appendix A. [[Omitted]]

[[Omitted in this draft.]]

Appendix B. Mandatory Attribute Descriptors for Distinguished Names

As per [RFC4514], attribute descriptors case-insensitive. A conformant implementation MUST recognize the attributes in the table below when parsing certspecs containing distinguished names, both by the OIDs and by the names recorded in the LDAP Parameters: Object

Identifier Descriptors registry [LDAPDESC]. A conforming generator SHOULD emit these attribute descriptors in lieu of their dotted decimal representations.

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]. Conforming implementations MAY generate this name from 2.5.4.42 and MUST parse this name as 2.5.4.42, despite its RESERVED status.

Table 1: Attribute Descriptors

Appendix C. Recommended Attribute Descriptors for issuersn certspec

As per [RFC4514], attribute descriptors case-insensitive. [[TODO: complete. Probably date of birth, place of birth, gender, etc. already defined elsewhere.]]

Appendix D. Algorithm for Distinguishing Between (Public Key) Certificates and Attribute Certificates

A certspec can identify a (public key) certificate ("PKC") or an attribute certificate ("AC"). When the type of certificate is specified unambiguously in the source data, an implementation SHOULD follow specifier in the source data. However, of the certspecs listed in this document, only a subset of URIs are capable of unambiguous specification (e.g., via Internet media type designation of application/pkix-cert or application/pkix-attr-cert). (The file extension ought not be considered a reliable indicator of the type.) Most other certspecs will return a blob of bytes or characters. Therefore, an implementation needs to implement some content-sniffing to figure out what the data represents. There are two (not entirely orthogonal) decisions: is the data textual [RFC7468] or binary (i.e., DER-encoded), and does the data represent a PKC or AC? (Note: The data-based certspecs BASE64 and HEX, always represent one DER-encoded certificate or ContentInfo/SignedData; the encodings MUST NOT encode a textual blob.)

This appendix provides an informative algorithm that implementations MAY use to do such content-sniffing.

Ensure that the first octet is SEQUENCE 30h.

If there are 2 elements -> confirm that the first element is OBJECT IDENTIFIER 1.2.840.113549.1.7.2 and the second element is explicitly

tagged (APPLICATION 0, A0). This is a ContentInfo containing SignedData.

Otherwise, ensure that there are 3 elements, and that the first element is a SEQUENCE 30h (either AttributeCertificateInfo or TBSCertificate).

this SEQUENCE has: 6, 7, or 7+ elements

if 6 elements -> V1 certificate

if 7+ elements ->

look at version filed (first element)

if INTEGER (UNIVERSAL 2), it's an attribute certificate

if explicitly tagged (APPLICATION 0, A0) and the contents are INTEGER (UNIVERSAL 2), it's a public key certificate.

otherwise -> malformed, not in DER.

END

If DER (or DER-like material, i.e., BER that an application chooses to accept as if it were DER anyway) is found, the complete certificate SHOULD be the only PDU in the data blob, and SHOULD occupy the entirety of the data blob. Pathological cases may exist, however, where data is appended to the end of the blob, that is not part of the certificate. An implementation has three choices: a) ignore the data, b) attempt to parse the data for additional certificates, c) reject the entire data blob as malformed (thus, rejecting at least the initial identified certificate). c) is valid behavior. a) depends on the nature of the identification (e.g., if the certspec is a hash, the application MUST confirm that the hash is computed over the actual certificate octets, and does not include the jetsam past the end of the certificate). If b) is attempted, the implementation MUST verify that the first certificate matches the identification (see comment on a)), and that subsequent certificates, if found, match the identification as well.

If the data blob is found not to contain DER (or DER-like material, see above), the data may be textual. [RFC7468] outlines the possibilities of PKIX structures that could be present in such text. After determining one or more appropriate encoding possibilities, an implementation MUST scan the entire textual blob and handle the possibility that multiple certificates might be present. The implementation MUST NOT stop at parsing the first one, the last one,

or some middle one after it "tires out". Parsers SHOULD [[NB: or MUST, or MUST NOT??]] treat the label on a textually encoded item as definitive; therefore, parsers SHOULD NOT need to [[NB: or MUST NOT, or MUST??]] process all textually encoded items. [[NB: compare with Content-Type, which this document considers definitive: mislabeled contents get punished.]]

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/