Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

S. Leonard
Penango, Inc.

March 13, 2017

# Textual Specification for Certificates and Attributes
# draft-seantek-certspec-11

## 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.

## 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 September 14, 2017.

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 and speed 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 containing a public key [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 "entity" means a MIME entity [RFC2045], namely, 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 so that someone glancing at a certspec can ascertain the data types that it comprises. This specification addresses the "speed" goal by incorporating identifiers that implementations typically use as indexes to certificate databases. For instance, many implementations index certificates by issuer and serial number, or by SHA-1 hash, regardless of how collision resistant those pieces of data are at present.

### 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

Certspecs and their attendant elements are textual strings, and are intended for use with textual protocols. Where possible, certspecs are just identifiers from other protocols with minimal syntactic sugar to distinguish one type of certspec from another. Several certspec productions look like URIs, but are not. To distinguish certspec 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. Basic Syntax and ABNF

The bulk of this document defines textual formats for interchange. While textual strings in this document can be in any character encoding, the delimiter characters in this document are drawn from ASCII. Unicode [UNICODE] support at some level (e.g., in attribute values that are in LDAP string form) is inevitable, but the general premise is that an implementation can convert the production (or appropriate parts of the production) to Unicode when it is needed. The ABNF in this document is normative, and is drawn from [RFC5234], [RFC7405], [I-D.seantek-abnf-more-core-rules], and [I-D.seantek-unicode-in-abnf]. The ABNF reuses certain definitions from [RFC4512] and [RFC4514], but does not formally reference those rules due to notating Unicode characters beyond the ASCII range in a more modern way.

# 4. certstring Syntax

A certificate string ("certstring") is a string with a single certspec (see Section 5) or multiple certspecs (a "multispec", see Section 8), followed by an optional set of attributes ("pkcsattrs", see Section 9). A multispec is a discriminator for a single certificate. In contrast, pkcsattrs are optional attributes associated with a single certificate. These attributes do not participate in selecting a certificate, but might be used to identify other things, such as the token on which associated private keying material resides. The string has the ABNF:

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

Figure 1: certstring ABNF

## 5. 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 ABNF is:

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

certspec-hash    = "SHA-1"   ":" 40HEXDIG /
                   "SHA-256" ":" 64HEXDIG /
                   "SHA-384" ":" 96HEXDIG /
                   "SHA-512" ":" 128HEXDIG

; Proposal: Hash Function Textual Name registry hereby limited
; to RFC 3986 scheme characters

certspec-content = ("HEX" / "BASE16") ":" 1*(2HEXDIG) /
                   "BASE64"  ":" base64string

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

; based on [RFC4512][RFC4514]
distinguishedName = [ relativeDistinguishedName
          *( "," relativeDistinguishedName ) ]
relativeDistinguishedName = attributeTypeAndValue
          *( "+" attributeTypeAndValue )
attributeTypeAndValue = attributeType "=" attributeValue
attributeType = descr / numericoid

num = "0" / %x31-39 *DIGIT
descr = ALPHA *(ALPHA / DIGIT / "-")
numericoid = ("0" / "1" / "2") 1*("." num)

attributeValue = string / hexstring
string      = [ ( leadchar / pair ) [ *( stringchar / pair )
              ( trailchar / pair ) ] ]
; excl SP " # + , ; < > \
leadchar    = %x01-1F / %x21 / %x24-2A / %x2D-3A /
              %x3D / %x3F-5B / %x5D-7F / BEYONDASCII
; excl SP "   + , ; < > \
```

```
   trailchar   = %x01-1F / %x21 / %x23-2A / %x2D-3A /
                 %x3D / %x3F-5B / %x5D-7F / BEYONDASCII
   ; excl    "   + , ; < > \
   stringchar  = %x01-21 / %x23-2A / %x2D-3A /
                 %x3D / %x3F-5B / %x5D-7F / BEYONDASCII

   pair        = "\" (" " / %x22-23 / %x2B-2C / %x3B-3E / "\" / 2HEXDIG)
   hexstring   = "#" 2HEXDIG

   certspec-el   = "ISSUERSN" ":" distinguishedName ";" serialNumber /
                   "SKI"      ":" 1*(2HEXDIG)

   serialNumber  = 1*(2HEXDIG)

   certspec-path     = certspec-uri / certspec-file / certspec-reg

   ; from RFC3986; RFC 6570
   ; superfluous: URI-reference     = URI-reference@[RFC3986]
   URI-Template      = URI-Template@[RFC6570]

   certspec-uri      = "URI:" URI-Template

   ; see POSIX, etc.
   certspec-file     = ("/" / "\" / [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

   certspec-reg      = reg-hive 1*("\" reg-key)
                       ["\\" reg-value-name]

   reg-hive          = reg-local-hive / reg-remote-hive
   reg-local-hive    = "HKEY_LOCAL_MACHINE" / "HKEY_CURRENT_USER" /
                       "HKEY_CLASSES_ROOT" / "HKEY_USERS" /
                       "HKEY_CURRENT_CONFIG" /
                       "HKLM:" / "HKCU:" / "HKCR:" /
                       "HKU:" / "HKCC:"
   ; TODO: better specify computer name; it could be a NETBIOS name...?
   reg-remote-hive   = "\\" reg-name@[RFC3986] "\" ("HKLM" / "HKU") ":"

   ; escape < > |. Need to figure out if 7F is ok, also C0, C1, etc.
```

```
reg-key          = 1*(%x20-3B / %x3D / %x3F-5B %x5D-7B / %x7D.7E /
                      "\<" / "\>" / "\|" / BEYONDASCII)
; reg-value-name can be empty
reg-value-name   = *(%x20-3B / %x3D / %x3F-7B / %x7D.7E /
                      "\<" / "\>" / "\|" / BEYONDASCII)
```

Figure 2: certspec ABNF

### 5.1. 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.

## 6. Standard Certificate Specifications

Standard certificate specifications are intended for interchange as user- and developer-friendly identifiers for individual certificates. This section provides four cryptographic hash-based certspecs, two content-based certspecs, two element-based certspecs, and three path-based certspecs.

### 6.1. 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.

#### 6.1.1. SHA-1

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

#### 6.1.2. SHA-256

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

#### 6.1.3. SHA-384

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

#### 6.1.4. SHA-512

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

### 6.2. 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) followed by any octet other than 80h (the marker for indefinite length encoding). See also Section 6.5.

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.

#### 6.2.1. BASE64

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

#### 6.2.2. 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.

### 6.3. 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.

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

#### 6.3.1. ISSUERSN: Issuer Name and Serial Number

The introducer production is "ISSUERSN:".

##### 6.3.1.1. 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.

##### 6.3.1.2. 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].

#### 6.3.1.3. 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 A. Implementations are encouraged to recognize additional attribute descriptors where possible. A sample list of such attribute descriptors is provided in Appendix B. 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.

### 6.3.2. 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.

## 6.4. Path-Based Specifications

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

Two common themes among path-based certspecs are that they may refer to weakly typed or untyped data, and they have a higher probability of referring to data that contains multiple certificates. Therefore, a greater degree of content-sniffing is required for interoperability than the certspecs above. An implementation that implements these

path-based certspecs SHALL support the ASN.1 Certificate PDU when public key certificates are being retrieved, and the ASN.1 AttributeCertificate PDU when attribute certificates are being retrieved. Additionally, a conforming implementation SHALL support the ASN.1 ContentInfo (PKCS #7/CMS SignedData) PDU.

Untyped binary data may be encoded in a [X.690] transfer syntax, which may be BER, CER, or DER; for purposes of this section, these are all called "BER-encoded".

### 6.4.1. 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 the filename, which SHOULD NOT be the conclusive determinant of the type:

1. Typed data (reported only by a minority of file systems), which is treated conclusively as the type

2. Data that is determined to be textual, which is analyzed according to [RFC7468]

3. Data that is determined to be BER-encoded

The manner of determining whether data is textual or BER-encoded data is not fixed by this specification, but see, e.g., Appendix C.

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 pkcsattrs delimiter. Windows-oriented file paths cannot contain "|".

### 6.4.2. Registry

Certificates can be identified on Windows machines with Registry keys and values. The introducer productions for local Registry entries are "HKEY_LOCAL_MACHINE\", "HKEY_CURRENT_USER\", "HKEY_CLASSES_ROOT\", "HKEY_USERS\", "HKEY_CURRENT_CONFIG\", "HKLM:\", "HKCU:\", "HKCR:\", "HKU:\", and "HKCC:\". The introducer productions for remote Registry entries are "\\", followed by a computer name, followed by either "\HKLM:\" or "\HKU:\".

Registry key names include any printable character except backslash "\"; each key includes what amounts to an associative array of values, which are name/type/data tuples. A value name can include any printable character, including "<", ">", and "|"; additionally, every key has a default value, which has a zero-length name. This layout presents a couple of parsing challenges.

A Registry production is comprised of a key path followed by a value. The key path's key names are delimited by "\". In each key name, "<", ">", and "|" SHALL be escaped with a preceding "\". The value name is delimited from the key name with two backslashes "\\". "<", ">", "|", and "\" in the value SHALL be escaped with a preceding "\". The default value MAY be identified with or without the two final backslashes. Unlike file paths, Registry productions do not recognize or substitute unexpanded environment variables.

Registry values have a type and some data. When the type is REG_SZ or REG_EXPAND_SZ, an implementation is to treat the text firstly as a recursive certspec or multispec. If it is not a certspec or multispec, then an implementation is to analyze the text according to [RFC7468]. Text in REG_EXPAND_SZ is subject to environment variable substitution. When the type is REG_BINARY, an implementation is to determine if the data is BER-encoded, and if so, to analyze it for supported ASN.1 PDUs. When the type is REG_LINK, an implementation is to follow the symbolic link.

### 6.4.3. URI

The introducer production is "URI:". The value is a URI-Template production [RFC6570], which is to produce a [RFC3986] conforming URI-reference production.

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. representations that are conclusively public key certificates or attribute certificates, such as LDAP URIs [RFC4516] that point to or contain userCertificate attributes (2.5.4.36, for public key certificates) or attributeCertificate attributes (2.5.4.58, for attribute certificates)

2. application/pkix-cert and application/pkix-attr-cert entities, which are conclusively public key certificates or attribute certificates, respectively

3. application/pkcs7-mime and application/cms entities, when the body represents a ContentInfo/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)

4. text/plain entities, which are analyzed according to [RFC7468]

5. Arbitrary data and application/octet-stream entities are treated as untyped; they are are analyzed for textual or binary [X.690] data

6. Arbitrary text, which is analyzed according to [RFC7468]

7. Arbitrary BER-encoded data, which is analyzed for supported ASN.1 PDUs

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 pkcsattrs delimiter.

### 6.5. Algorithm for Distinguishing ASN.1 PDUs

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 SHALL

follow the 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). Most other certspecs will return a blob of bytes or characters. Therefore, an implementation needs to perform some content-sniffing to figure out what the data represents. There are two (not entirely orthogonal) decisions: is the data textual [RFC7468] or not, and does the data represent a PKC or AC? (Note: The content-based certspecs BASE64 and HEX always represent one certificate; the encodings MUST NOT encode a textual blob or a PKCS #7/CMS PDU.)

This normative section addresses distinguishing PDU types when applications encounter BER-encoded data that is not further typed. An implementation MAY use any algorithm it chooses, as long as it produces the same results. A suggested algorithm for distinguishing textual data is in Appendix C; that algorithm is merely informative.

The algorithm for distinguishing ASN.1 PDUs is:

1. Ensure that the first octet is SEQUENCE 30h.

2. Ensure that the length covers the length of the data, minus the tag and length octets. (If the length is indefinite, a check that the end of the data has the end-of-contents octets would be appropriate.) Extraneous data SHALL be considered erroneous.

3. 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). Analyze the PDU as a ContentInfo containing SignedData.

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

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

6. if 6 elements -> Analyze the PDU as a Certificate (public key certificate) with version v1 (ABSENT).

7. if 7+ elements ->

    1. look at version field (first element)

    2. if INTEGER (UNIVERSAL 2) -> Analyze the PDU as an AttributeCertificate (attribute certificate)

3. if explicitly tagged (APPLICATION 0, A0) and the contents are INTEGER (UNIVERSAL 2) -> Analyze the PDU as a Certificate (public key certificate).

## 7. 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.

### 7.1. 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.

### 7.2. 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.

## 8. 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 multispec 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.

Another example is to provide two URIs for a certificate: one that works inside an organizational firewall, and one that works outside an organizational firewall. Conforming applications MAY ignore individual certspec lookup failures (where the certspec fails to return any certificate due to error conditions) as environment, capability, and security concerns dictate.

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

## 9. Attributes (pkcsattrs)

This specification defines a textual format for PKCS attributes. This format is not limited to certificates: it can be used with other PKCS-related data. The syntax is intended primarily to convey certificate-related attributes found in PKCS #9 [RFC2985], PKCS #11 [PKCS11], PKCS #12 [RFC7292], and particular implementations of cryptographic libraries. These attributes are syntactically identical to, but semantically disjoint from, Directory (X.500/LDAP) attributes.

When pkcsattrs is used with a certspec or multispec, the intent is to associate arbitrary metadata with a certificate--metadata that is not intrinsic to that certificate. For example, the additional attributes may represent preferences. Attributes in this context are semantically equivalent to PKCS #12 "bagAttributes", drawn from the "PKCS12AttrSet" [RFC7292].

pkcsattrs are delimited from a certspec or multispec production with "|". Each pkcsattr SHALL have a corresponding ASN.1 definition. The textual syntax of pkcsattrs is very similar to (in fact, a superset of) [RFC4514]: the pkcsattrs 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 MUST support parsing through this encoding, but merely MAY support this encoding (encoding and decoding between [X.690]) for attributes of interest to it.

ASN.1 value: The attrValue can be represented by its ASN.1 value notation [X.680], surrounded by exactly one space (SP) on each end. The syntax is precisely defined in Figure 4 so that the value itself never begins or ends with ASN.1 "white-space", although "white-space" can occur within the value. 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 lists (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 MUST support parsing through this encoding, but merely MAY support this encoding (encoding and decoding between [X.690]) for attributes of interest to it.

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 pkcsattrs. The other

representations are really meant for human production and consumption.

## 9.1. ABNF

The collective ABNF of pkcsattrs is:

```
pkcsattrs = pkcsattr ["," pkcsattrs]
pkcsattr  = pkcsattrType ["=" pkcsattrValues]
pkcsattrType = descr / numericoid
pkcsattrValues = pkcsattrValue ["+" pkcsAttrValues]
pkcsattrValue = hexstring / string /
                basic-xer-string / asn1-value-string

basic-xer-string = xer-element

; limited by [X.680][X.693]
xer-Name = ALPHA *(ALPHA / DIGIT / "_" / "-" / ".")

; limited by XML to these four chars
xW = *(HT / LF / CR / SP)

xer-element = xer-EmptyElemTag / xer-STag xer-content xer-ETag

xer-EmptyElemTag = "<" xer-Name xW "/>"

xer-STag = "<" xer-Name xW ">"

xer-content = *xer-CharData *((xer-element / xer-Reference)
              *xer-CharData)

xer-ETag = "</" xer-Name xW ">"

xer-CharData = HT / LF / CR / %x20-25 / %x27-3B / "=" / %x3F-D7FF /
               %xE000-%xFFFD / %x10000-10FFFF

xer-Reference = xer-EntityRef / xer-CharRef

xer-EntityRef = "&" (%s"amp" / %s"lt" / %s"gt") ";"

xer-CharRef = "&#" (1*DIGIT / %s"x" 1*HEXDIG) ";"

; TODO: may want another delimiter--think about it
; uses num from above for non-negative integers
asn1-value-string = SP aValue aW

; identifier, Clause 12.3 of [X.680]
aid = %x61-7A *(["-"] (ALPHA / DIGIT))
```

```
; "newline", Clause 12.1.6 of [X.680]
; (NEL LS PS omitted)
aNL = %d10-13
; single "white-space" (comment considered matching,
; because delimits lexical items), Clause 12.1.6 of [X.680]
aS = %d9-13 / SP / NBSP / acomment
aW = *aS

acomment = "--" *(["-"] ( UWSP / %x21-2C / %x2E-7E /
                          UVCHARBEYONDASCII / PUACHAR ) )
               (aNL / "-" (aNL / "-"))

; space in unicode: 85 A0 1680 2000-200A 202F 205F 3000
; related not-unicode-White_Space-but-whitespace
;  180E 200B-200D 2060 FEFF

; uses "white-space" above, but "comment" not relevant
acstring = %x22 *(UWSP / aNL / %x21 / %x22.x22 / %x23-7E /
           UVCHARBEYONDASCII) %x22

aValue = %s"TRUE" / %s"FALSE" / %s"NULL" / %s"PLUS-INFINITY" /
     %s"MINUS-INFINITY" / %s"NOT-A-NUMBER" /
     "'" *("0" / "1" / aW) "'B" / "'" *(HEXDIG / aW) "'H" /
     %s"CONTAINING" 1*aS aValue /
     aid [aW ":" aW aValue] /
     aNumericRealValue / acstring / "{" aW alist "}"

; ObjIdComponents [X.680]
; aOIDC
aObjIdComponents = (num / aid) [aW "(" aW (num/aid) aW ")"]

; NumericRealValue [X.680]
; TODO: integer part could be 1*DIGIT or num
aNumericRealValue = ["-" aW] 1*DIGIT ["."] *DIGIT ["e" ["-"] num]

; *List values [X.680]
alist = aValue aW          *("," aW aValue aW)          /
        aid 1*aS aValue aW *("," aW aid 1*aS aValue aW) /
;       aOIDC              *(1*aS aOIDC)              aW
        aObjIdComponents   *(1*aS aObjIdComponents)  aW
```

Figure 4: pkcsattrs ABNF

## 9.2. Mandatory Attribute Support

### 9.2.1. In General

Attributes related to certificate objects are in the domain of PKCS attributes, not Directory name attributes. [I-D.seantek-ldap-pkcs9] discusses the problem and registers attributes that are specifically designated for PKCS use, rather than Directory use.

A conforming implementation is expected to recognize the short names (descriptors) recorded in the LDAP Parameters: Object Identifier Descriptors registry [LDAPDESC] that are designated for PKCS use if the implementation processes that attribute. Not all attributes are needed by all implementations. For example, a CMS processing application that supports pkcsattrs needs to recognize contentType and messageDigest, but not extendedCertificateAttributes.

### 9.2.2. For Certificate Applications

A conforming implementation that supports pkcsattrs for certificates SHALL process the following attributes from PKCS #9 [RFC2985], including recognizing the following short names (descriptors) and associated LDAP-specific string encodings.

friendlyName (1.2.840.113549.1.9.20)

localKeyId (1.2.840.113549.1.9.21)

signingDescription (1.2.840.113549.1.9.13)

smimeCapabilities (1.2.840.113549.1.9.15)
[[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.]]

## 9.3. Canonicalization

The pkcsattrs 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.

# 10. 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, pkcsattrs, 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 pkcsattrs 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 pkcsattrType production in pkcsattrs.

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.

## 11. 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. For security reasons, the introducers "MD2" and "MD5" SHALL NOT be generated or parsed. See [RFC6149] and [RFC6151].

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).

The ISSUERSN, SUBJECTEXP, and HOLDEREXP certspecs provide opportunities to identify the issuer, subject, or holder using multiple methods. An implementation MAY support other productions that equate to issuer certificates, subject identifiers, or holder sub-fields.

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 and Registry keys and values 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 pkcsattrs (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.

## 12. 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.

## 13. IANA Considerations

Appendix D proposes modifications to the application/pkcs12 media type to support labeling a degenerate syntax that only contains certificates and certificate revocation lists. IANA is asked to update the fields of the application/pkcs12 registration as follows:

```
Optional parameters:
 profile: A profile of PKCS #12 for particular applications.
  When this parameter has value "certs-only", then it conforms
  to the profile in Appendix E of [[RFC Ed: this document]].
    If a filename is supplied, the file extension is to be .p12c;
  appropriate description strings (in US-English) might be
    "PKCS #12 Certificate Store" or
    "PKCS #12 Certificate Data with Attributes", among others.
  It would be inappropriate to imply that such content
  contains keys or other secret materials.

  This parameter is case sensitive.

Published specification:
PKCS #12 v1.0, June 1999; PKCS #12 v1.1 (RFC 7292), July 2014
[[RFC Ed: add reference to this document.]]

Additional information:

  Deprecated alias names for this type: N/A
  Magic number(s): None.
  File extension(s): .p12 or .pfx; .p12c (in profile=certs-only case)
  Macintosh file type code(s): N/A
```

Figure 5: application/pkcs12 Media Type Registration

Appendix D proposes modifications to the application/cms media type to support SafeContents as a CMS inner content type. IANA is asked to update the CMS Inner Content Types sub-registry by adding an identifier "safeContents" with the object identifier listed in Appendix D. IANA is further asked to update the application/cms media type registration template accordingly.

## 14. 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 MD2 [RFC6149] and 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.

Conveying PKCS attributes with certificates will likely have security effects. For example, some implementations display the friendlyName attribute to a user on par with or in lieu of data derived from the certificate itself. Other implementations allow certificates to be

identified by this friendlyName attribute. Therefore, blind acceptance of PKCS attributes without considering the source or content can result in security compromises.

## 15. References

### 15.1. Normative References

[I-D.seantek-abnf-more-core-rules]
Leonard, S., "Comprehensive Core Rules and References for ABNF", draft-seantek-abnf-more-core-rules-06 (work in progress), September 2016.

[I-D.seantek-ldap-pkcs9]
Leonard, S., "Lightweight Directory Access Protocol (LDAP) Registrations for PKCS #9", draft-seantek-ldap-pkcs9-05 (work in progress), June 2016.

[I-D.seantek-unicode-in-abnf]
Leonard, S. and P. Kyzivat, "Unicode in ABNF", draft-seantek-unicode-in-abnf-00 (work in progress), September 2016.

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

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

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

[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.

[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>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <http://www.rfc-editor.org/info/rfc7405>.

[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>.

## 15.2. Informative References

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

[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.

[RFC6149] Turner, S. and L. Chen, "MD2 to Historic Status", RFC 6149, DOI 10.17487/RFC6149, March 2011, <http://www.rfc-editor.org/info/rfc6149>.

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

[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.501] International Telecommunication Union, "Information technology - Open Systems Interconnection - The Directory: Models", ITU-T Recommendation X.501, ISO/IEC 9594-2, October 2012, <https://itu.int/ITU-T/X.501>.

[X.680] International Telecommunication Union, "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1, August 2015, <https://itu.int/ITU-T/X.680>.

[X.690] International Telecommunication Union, "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1, August 2015, <https://itu.int/ITU-T/X.690>.

[X.693] International Telecommunication Union, "Information technology - ASN.1 encoding rules: XML Encoding Rules (XER)", ITU-T Recommendation X.693, ISO/IEC 8825-4, August 2015, <https://itu.int/ITU-T/X.693>.

## Appendix A. 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)<br>commonName | 4514 |
| 2.5.4.7 | l (L)<br>localityName | 4514 |
| 2.5.4.8 | st (ST)<br>(S)*<br>stateOrProvinceName | 4514 |
| 2.5.4.10 | o (O)<br>organizationName | 4514 |
| 2.5.4.11 | ou (OU)<br>organizationalUnitName | 4514 |
| 2.5.4.6 | c (C)<br>countryName | 4514 |
| 2.5.4.9 | street (STREET)<br>streetAddress | 4514 |
| 0.9.2342.19200300.100.1.25 | dc (DC)<br>domainComponent | 4514 |

| | | |
|---|---|---|
| 0.9.2342.19200300.100.1.1 | uid (UID)<br>userId | 4514 |
| 2.5.4.5 | serialNumber (SERIALNUMBER) | 5280 |
| 2.5.4.46 | dnQualifier (DNQUALIFIER) | 5280 |
| 2.5.4.4 | sn (SN)<br>surname | 5280 |
| 2.5.4.42 | gn (GN)**<br>givenName | 5280 |
| 2.5.4.12 | (T)*<br>title | 5280 |
| 2.5.4.43 | (I)*<br>initials | 5280 |
| 2.5.4.44 | (GENQUALIFIER)*<br>generationQualifier<br>(GENERATIONQUALIFIER) | 5280 |
| 2.5.4.65 | (PNYM)*<br>pseudonym (PSEUDONYM) | 5280 |
| 1.2.840.113549.1.9.1 | (E)*<br>emailAddress<br>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 B. Recommended Attribute Descriptors for issuersn certspec

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

## Appendix C. Suggested Algorithm for Distinguishing Textual Data

This appendix provides an informative algorithm that implementations MAY use to content-sniff textual data. This appendix is a companion to Section 6.5.

Some certspecs will return arbitrary data, which might be textual in nature. This is especially true of the file path and URI specs. There are historical reasons for this, mostly boiling down to "DER" vs. "PEM" output options in popular cryptographic software packages, without clear guidance on file extensions.

The only BER-encoded PDUs that are mandated by this spec are Certificate [RFC5280], AttributeCertificate [RFC5755], and ContentInfo (containing SignedData) [RFC5652]. Before trying to do charset-sniffing, it is reasonable to probe for BER decoding first to see what happens. The (normative) algorithm in Section 6.5 is a sufficient test. At the very least, an implementation ought to check for the presence of the SEQUENCE octet (30h), a valid length that covers the length of the data, minus the tag and length octets, and two or three validly-encoded tag-length-value elements (Clause 8.1.1 of [X.690]) inside the SEQUENCE.

Although an implementation can ingest arbitrary text containing certificates, this specification only requires [RFC7468], which requires the presence of encapsulation boundaries. An implementation ought to look for Unicode byte order marks first; failing that, it ought to consider ASCII, basically ignoring invalid byte sequences that do not appear in [RFC7468] productions. Regardless of the charset(s) chosen, an implementation can hunt for the minimum string "-----BEGIN " followed somewhere by "-----END ", since those strings are required for [RFC7468] conformance.

## Appendix D. Binary Formats for Conveying Certificates with Attributes

During the development of this document, the author noted that there is a lack of standardization around conveying attributes with certificates. The bulk of this specification can be used to convey such attributes in text; however, a binary format is also desirable. Because the attributes in Section 9 are semantically equivalent to PKCS #12 "bagAttributes", it makes sense to reuse this structure of PKCS #12, if possible.

### D.1. PKCS #12 certs-only Profile

Predecessors to PKCS #12 have been criticized for being too obtuse and cumbersome to implement. This section proposes a profile of PKCS #12 for a degenerate case of the syntax that only conveys certificates and certificate revocation lists. It is analogous to the degenerate case of SignedData in CMS [RFC5652]. The overall usability goal is to convey certificates with attributes without requiring user input of secret or private material (i.e., a password or private key) to receive the data. The data MAY be signed for

integrity protection, so long as verifying the signature does not require user input of secret or private material.

To compose the degenerate case, the following structures in the "PFX" structure are limited:

1. authSafe has contentType id-data or id-signedData (if signed), containing an AuthenticatedSafe.

2. The AuthenticatedSafe SHALL contain exactly one ContentInfo, which has contentType id-data, containing a SafeContents.

3. The SafeContents can contain any number of SafeBags.

4. Each SafeBag can only contain a certificate (via certBag) or a certificate revocation list (via crlBag).

5. macData SHALL be "ABSENT".

[RFC7292] does not have an identifier for attribute certificates in the CertBag. The ASN.1 module is hereby modified to support attribute certificates:

```
attributeCertificate BAG-TYPE ::=
    {OCTET STRING IDENTIFIED BY {certTypes 3}} -- 3 is TBD
    -- DER-encoded attribute certificate stored in OCTET STRING

CertTypes BAG-TYPE ::= {
    x509Certificate |
    sdsiCertificate,
    ...,
    attributeCertificate,
    ... }
```

Figure 6: PKCS #12 ASN.1 Module Modification

Implementations MUST parse through certBag elements containing attribute certificates (MUST NOT fail parsing), but actually processing attribute certificates is OPTIONAL if an implementation has no need for them. (The same remarks apply to certificate revocation lists.) Because this profile does not use encryption or key derivation functions, conforming implementations do not need to support such algorithms, which should greatly simplify implementations.

It is desirable to convey this degenerate PKCS #12 data in MIME entities and files. Since this format has very different usability properties from full-featured PKCS #12, it is not to be labeled as

standard PKCS #12. A new "profile" optional parameter with value "certs-only" is proposed for the application/pkcs12 media type, as well as a new file extension .p12c. See Section 13 for the modified registration template.

### D.2. CMS SafeContents contentType

Some applications will not want to bother with the PFX PDU of PKCS #12 at all. For those applications, it is possible to transmit SafeContents directly as CMS (PKCS #7) content.

The CMS (TBD: or PKCS #12?) ASN.1 module is hereby enhanced to include an object identifier for SafeContents as a content type:

```
IMPORTS
SafeContents, bagtypes
FROM PKCS-12 {1 2 840 113549 1 pkcs-12(12) modules(0) pkcs-12(1)}


ContentSet CONTENT-TYPE ::= {
    --  Define the set of content types to be recognized.
    ct-Data | ct-SignedData | ct-EncryptedData | ct-EnvelopedData |
    ct-AuthenticatedData | ct-DigestedData | ct-SafeContents, ... }

ct-SafeContents CONTENT-TYPE ::=
    { SafeContents IDENTIFIED BY id-safeContents }

-- could be 1.2.840.113549.1.12.10.1.6 existing safeContentsBag OID,
-- or a new 1.2.840.113549.1.12.10.2,
-- or a new 1.2.840.113549.7.9 from PKCS #7,
-- or a new 1.2.840.113549.1.9.16.1.37 from pkcs-9 smime ct
id-safeContents OBJECT IDENTIFIER ::= {bagtypes 6}
```

Figure 7: CMS ASN.1 Module Modification

SafeContents is registered as a CMS Inner Content Type (ICT) with the identifier "safeContents". See Section 13 for the relevant registration and application/cms modification.

Using this technique allows SafeContents directly in CMS content. Any kind of SafeBag is permitted inside; unlike Appendix D.1, this format is not further profiled.

### D.3. SafeContents-to-PKCS#12 BER Adapter

An application that processes a SafeContents PDU directly may find it expedient to adapt it to a PFX PDU for ingestion into legacy code that only processes PKCS #12 data. The following adapter can be used

for that purpose. Octets are listed in hexadecimal. Place the BER encoding of SafeContents in the position marked "(SafeContents)". The placeholders "LEN-" are [X.690] length octets, which are to be computed as follows:

LEN-SC: The length in octets of the SafeContents.

LEN-2: LEN-SC plus the length in octets of LEN-SC itself, plus 1.

LEN-3: LEN-2 plus the length in octets of LEN-2 itself, plus 20.

LEN-4: LEN-3 plus the length in octets of LEN-3 itself, plus 1.

```
30 80 02 01 03 30 80 06092A864886F70D010701 A0 LEN-4 04 LEN-3
30 80 30 80 06092A864886F70D010701 A0 LEN-2 04 LEN-SC
(SafeContents)
0000 0000 0000 0000
```

Figure 8: BER Adapter

## Appendix E. Textual Encoding of Attributes

[[TODO: Consider removing this appendix; Section 9 is clearer.]]

From time to time, it is desirable to convey attributes independently of other PKIX, PKCS, or CMS structures. This appendix defines a textual encoding [RFC7468] format for attributes.

Attributes are encoded using the "ATTRIBUTES" label. The encoded data MUST be a BER (DER strongly preferred; see Appendix B of [RFC7468]) encoded ASN.1 "SET OF Attribute", or, in rare cases, "SEQUENCE OF Attribute" structure as described throughout Directory, PKIX, PKCS, and CMS standards. Unless the collection is specifically ordered, emitting the "SET OF Attribute" variant is RECOMMENDED.

No IETF document formally discusses what an attribute is (although [RFC4512] comes close). Workable definitions can be gleaned from [X.501] and [RFC4512]:

> Each attribute [in the Directory] provides a piece of information about, or describes a particular characteristic of, the object to which the entry corresponds. --Clause 8.2 of [X.501]
>
> An attribute consists of an _attribute type_, which identifies the class of information given by an attribute, and the corresponding _attribute values_, which are the particular instances of that class of information appearing in the attribute within the entry. --Clause 8.2 of [X.501]

> An entry [in the Directory] consists of a set of attributes that hold information about the object that the entry represents. --Section 2.2 of [RFC4512]

> An attribute is an attribute description (a type and zero or more options) with one or more associated values. An attribute is often referred to by its attribute description. --Section 2.2 of [RFC4512]

An attribute is comprised of a type and a SET OF values. The collection of values is always unordered. Collections of attributes are almost always unordered, and are almost always stored in a "SET OF Attribute". A few protocols store attributes in a "SEQUENCE OF Attribute", but for nearly all cases, the ordering is stated to be irrelevant by the relevant standard document.

The attribute type is a widely shared protocol element in LDAP/ Directory, PKIX, PKCS, and CMS standards. However, the collections of relevant attributes to particular occurrences of the structure (as represented by a table constraint on an occurence) are largely disjoint from one another. CMS attribute collections (e.g., "SignedAttributes", "UnsignedAttributes", "UnprotectedAttributes") share no common semantics with Directory attributes, for instance. The textual encoding provided in this section is appropriate for any collection of attributes, but only context can determine what kinds of attributes are appropriate, as well as the identity of the corresponding object. Figure 9 provides an example of the textual encoding, along with its corresponding Section 9 format.

```
localKeyId=#0402534C,friendlyName=Chubby\F0\9F\90\B0
-----BEGIN ATTRIBUTES-----
MTQwEQYJKoZIhvcNAQkVMQQEAlNMMB8GCSqGSIb3DQEJFDESHhAAQwBoAHUAYgBi
AHnYPdww
-----END ATTRIBUTES-----
```

Figure 9: Attributes Example

## 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/