Textual Encodings of PKIX, PKCS, and CMS Structures

Abstract

This document describes and discusses the textual encodings of the Public-Key Infrastructure X.509 (PKIX), Public-Key Cryptography Standards (PKCS), and Cryptographic Message Syntax (CMS). The textual encodings are well-known, are implemented by several applications and libraries, and are widely deployed. This document articulates the de facto rules by which existing implementations operate and defines them so that future implementations can interoperate.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback is obtained at http://www.rfc-editor.org/info/rfc7468.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

Several security-related standards used on the Internet define ASN.1 data formats that are normally encoded using the Basic Encoding Rules (BER) or Distinguished Encoding Rules (DER) [X.690], which are binary, octet-oriented encodings. This document is about the textual encodings of the following formats:

1. Certificates, Certificate Revocation Lists (CRLs), and Subject Public Key Info structures in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile [RFC5280].
2. PKCS #10: Certification Request Syntax [RFC2986].
3. PKCS #7: Cryptographic Message Syntax [RFC2315].
5. PKCS #8: Private-Key Information Syntax [RFC5208], renamed to One Asymmetric Key in Asymmetric Key Package [RFC5958], and Encrypted Private-Key Information Syntax in the same documents.


A disadvantage of a binary data format is that it cannot be interchanged in textual transports, such as email or text documents. One advantage with text-based encodings is that they are easy to modify using common text editors; for example, a user may concatenate several certificates to form a certificate chain with copy-and-paste operations.

The tradition within the RFC series can be traced back to Privacy-Enhanced Mail (PEM) [RFC1421], based on a proposal by Marshall Rose in Message Encapsulation [RFC934]. Originally called "PEM encapsulation mechanism", "encapsulated PEM message", or (arguably) "PEM printable encoding", today the format is sometimes referred to as "PEM encoding". Variations include OpenPGP ASCII armor [RFC4880] and OpenSSH key file format [RFC4716].

For reasons that basically boil down to non-coordination or inattention, many PKIX, PKCS, and CMS libraries implement a text-based encoding that is similar to -- but not identical with -- PEM encoding. This document specifies the textual encoding format, articulates the de facto rules that most implementations operate by, and provides recommendations that will promote interoperability going forward. This document also provides common nomenclature for syntax elements, reflecting the evolution of this de facto standard format. Peter Gutmann’s "X.509 Style Guide" [X.509SG] contains a section "base64 Encoding" that describes the formats and contains suggestions similar to what is in this document. All figures are real, functional examples, with key lengths and inner contents chosen to be as small as practicable.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. General Considerations

Textual encoding begins with a line comprising "-----BEGIN ", a label, and "-----", and ends with a line comprising "-----END ", a label, and "-----". Between these lines, or "encapsulation boundaries", are base64-encoded data according to Section 4 of [RFC4648]. (PEM [RFC1421] referred to this data as the "encapsulated
text portion"). Data before the encapsulation boundaries are permitted, and parsers MUST NOT malfunction when processing such data. Furthermore, parsers SHOULD ignore whitespace and other non-base64 characters and MUST handle different newline conventions.

The type of data encoded is labeled depending on the type label in the "-----BEGIN " line (pre-encapsulation boundary). For example, the line may be "-----BEGIN CERTIFICATE-----" to indicate that the content is a PKIX certificate (see further below). Generators MUST put the same label on the "-----END " line (post-encapsulation boundary) as the corresponding "-----BEGIN " line. Labels are formally case-sensitive, uppercase, and comprised of zero or more characters; they do not contain consecutive spaces or hyphen-minuses, nor do they contain spaces or hyphen-minuses at either end. Parsers MAY disregard the label in the post-encapsulation boundary instead of signaling an error if there is a label mismatch: some extant implementations require the labels to match; others do not.

There is exactly one space character (SP) separating the "BEGIN" or "END" from the label. There are exactly five hyphen-minus (also known as dash) characters ("-") on both ends of the encapsulation boundaries, no more, no less.

The label type implies that the encoded data follows the specified syntax. Parsers MUST handle non-conforming data gracefully. However, not all parsers or generators prior to this document behave consistently. A conforming parser MAY interpret the contents as another label type but ought to be aware of the security implications discussed in the Security Considerations section. The labels described in this document identify container formats that are not specific to any particular cryptographic algorithm, a property consistent with algorithm agility. These formats use the ASN.1 AlgorithmIdentifier structure as described in Section 4.1.1.2 of [RFC5280].

Unlike legacy PEM encoding [RFC1421], OpenPGP ASCII armor, and the OpenSSH key file format, textual encoding does *not* define or permit headers to be encoded alongside the data. Empty space can appear between the pre-encapsulation boundary and the base64, but generators SHOULD NOT emit such any such spacing. (The provision for this empty area is a throwback to PEM, which defined an "encapsulated header portion".)

Implementers need to be aware that extant parsers diverge considerably on the handling of whitespace. In this document, "whitespace" means any character or series of characters that represent horizontal or vertical space in typography. In US-ASCII, whitespace means HT (0x09), VT (0x0B), FF (0x0C), SP (0x20), CR
(0x0D), and LF (0x0A); "blank" means HT and SP; lines are divided
with CRLF, CR, or LF. The common ABNF production WSP is congruent
with "blank"; a new production W is used for "whitespace". The ABNF
in Section 3 is specific to US-ASCII. As these textual encodings
can be used on many different systems as well as on long-term archival
storage media such as paper or engravings, an implementer ought to
use the spirit rather than the letter of the rules when generating or
parsing these formats in environments that are not strictly limited
to US-ASCII.

Most extant parsers ignore blanks at the ends of lines; blanks at the
beginnings of lines or in the middle of the base64-encoded data are
far less compatible. These observations are codified in Figure 1.
The most lax parser implementations are not line-oriented at all and
will accept any mixture of whitespace outside of the encapsulation
boundaries (see Figure 2). Such lax parsing may run the risk of
accepting text that was not intended to be accepted in the first
place (e.g., because the text was a snippet or sample).

Generators MUST wrap the base64-encoded lines so that each line
consists of exactly 64 characters except for the final line, which
will encode the remainder of the data (within the 64-character line
boundary), and they MUST NOT emit extraneous whitespace. Parsers MAY
handle other line sizes. These requirements are consistent with PEM
[RFC1421].

Files MAY contain multiple textual encoding instances. This is used,
for example, when a file contains several certificates. Whether the
instances are ordered or unordered depends on the context.

3. ABNF

The ABNF [RFC5234] of the textual encoding is:

textualmsg = preeb *WSP eol
              *eolWSP
              base64text
              posteb *WSP [eol]

preeb       = "-----BEGIN " label "-----" ; unlike [RFC1421] (A)BNF,
                      ; eol is not required (but
posteb       = "-----END " label "-----"   ; see [RFC1421], Section 4.4

base64char  = ALPHA / DIGIT / "+" / "/"

base64pad    = "="

base64line  = 1*base64char *WSP eol
base64finl = *base64char (base64pad *WSP eol base64pad /
   *2base64pad) *WSP eol
   ; ...AB= <EOL> = <EOL> is not good, but is valid

base64text = *base64line base64finl
   ; we could also use <encbinbody> from RFC 1421, which requires
   ; 16 groups of 4 chars, which means exactly 64 chars per
   ; line, except the final line, but this is more accurate

labelchar = %x21-2C / %x2E-7E    ; any printable character,
   ; except hyphen-minus

label      = [ labelchar *( [-" / SP] labelchar ) ]       ; empty ok

eol        = CRLF / CR / LF

eolWSP     = WSP / CR / LF                        ; compare with LWSP

Figure 1: ABNF (Standard)

laxtextualmsg = *W preeb
   laxbase64text
   posteb *W

W = WSP / CR / LF / %x0B / %x0C           ; whitespace

laxbase64text = *(W / base64char) [base64pad *W [base64pad *W]]

Figure 2: ABNF (Lax)

stricttextualmsg = preeb eol
   strictbase64text
   posteb eol

strictbase64finl = *15(4base64char) (4base64char / 3base64char
   base64pad / 2base64char 2base64pad) eol

base64fullline   = 64base64char eol

strictbase64text = *base64fullline strictbase64finl

Figure 3: ABNF (Strict)

New implementations SHOULD emit the strict format (Figure 3)
specified above. The choice of parsing strategy depends on the
context of use.
4. Guide

For convenience, these figures summarize the structures, encodings, and references in the following sections:

<table>
<thead>
<tr>
<th>Sec.</th>
<th>Label</th>
<th>ASN.1 Type</th>
<th>Reference Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>CERTIFICATE</td>
<td>Certificate</td>
<td>[RFC5280] id-pkix1-e</td>
</tr>
<tr>
<td>6</td>
<td>X509 CRL</td>
<td>CertificateList</td>
<td>[RFC5280] id-pkix1-e</td>
</tr>
<tr>
<td>7</td>
<td>CERTIFICATE REQUEST</td>
<td>CertificationRequest</td>
<td>[RFC2986] id-pkcs10</td>
</tr>
<tr>
<td>8</td>
<td>PKCS7</td>
<td>ContentInfo</td>
<td>[RFC2315] id-pkcs7*</td>
</tr>
<tr>
<td>9</td>
<td>CMS</td>
<td>ContentInfo</td>
<td>[RFC5652] id-cms2004</td>
</tr>
<tr>
<td>10</td>
<td>PRIVATE KEY</td>
<td>PrivateKeyInfo ::=</td>
<td>[RFC5208] id-pkcs8</td>
</tr>
<tr>
<td></td>
<td>OneAsymmetricKey</td>
<td></td>
<td>id-aKPV1</td>
</tr>
<tr>
<td>11</td>
<td>ENCRYPTED PRIVATE KEY</td>
<td>EncryptedPrivateKeyInfo</td>
<td>[RFC5958] id-aKPV1</td>
</tr>
<tr>
<td>12</td>
<td>ATTRIBUTE CERTIFICATE</td>
<td>AttributeCertificate</td>
<td>[RFC5755] id-acv2</td>
</tr>
<tr>
<td>13</td>
<td>PUBLIC KEY</td>
<td>SubjectPublicKeyInfo</td>
<td>[RFC5280] id-pkix1-e</td>
</tr>
</tbody>
</table>

Figure 4: Convenience Guide

id-pkixmod OBJECT IDENTIFIER ::= {iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) mod(0)}

id-pkix1-e OBJECT IDENTIFIER ::= {id-pkixmod pkix1-explicit(18)}

id-acv2 OBJECT IDENTIFIER ::= {id-pkixmod mod-attribute-cert-v2(61)}

id-pkcs OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)}

id-pkcs10 OBJECT IDENTIFIER ::= {id-pkcs 10 modules(1) pkcs-10(1)}

id-pkcs7 OBJECT IDENTIFIER ::= {id-pkcs 7 modules(0) pkcs-7(1)}

id-pkcs8 OBJECT IDENTIFIER ::= {id-pkcs 8 modules(1) pkcs-8(1)}

id-sm-mod OBJECT IDENTIFIER ::= {id-pkcs 9 smime(16) modules(0)}

id-aKPV1 OBJECT IDENTIFIER ::= {id-sm-mod mod-asymmetricKeyPkgV1(50)}

id-cms2004 OBJECT IDENTIFIER ::= {id-sm-mod cms-2004(24)}

* This OID does not actually appear in PKCS #7 v1.5 [RFC2315]. It was defined in the ASN.1 module to PKCS #7 v1.6 [P7v1.6], and has been carried forward through PKCS #12 [RFC7292].

Figure 5: ASN.1 Module Object Identifier Value Assignments
5. Textual Encoding of Certificates

5.1. Encoding

Public-key certificates are encoded using the "CERTIFICATE" label. The encoded data MUST be a BER (DER strongly preferred; see Appendix B) encoded ASN.1 Certificate structure as described in Section 4 of [RFC5280].

-----BEGIN CERTIFICATE-----
MIICLDCCAdKgAwIBAgIBADAKBggqhkjOPQQDAjB9MQswCQYDVQQGEwJCRTEPMA0GA1UE
ChMGR251VExTMSUwIwYDVQQIEwJCRTEPMA0GAIUEchMGR251VExTMSUwIwYDVQQIEw
JCRTEPMA0GAIUEchMGR251VExTMSUwIwYDVQQIEwJCRTEPMA0GAIUEchMGR251VEx
-----END CERTIFICATE-----

Figure 6: Certificate Example

Historically, the label "X509 CERTIFICATE" and also less commonly "X.509 CERTIFICATE" have been used. Generators conforming to this document MUST generate "CERTIFICATE" labels and MUST NOT generate "X509 CERTIFICATE" or "X.509 CERTIFICATE" labels. Parsers SHOULD NOT treat "X509 CERTIFICATE" or "X.509 CERTIFICATE" as equivalent to "CERTIFICATE", but a valid exception may be for backwards compatibility (potentially together with a warning).
5.2. Explanatory Text

Many tools are known to emit explanatory text before the BEGIN and after the END lines for PKIX certificates, more than any other type. If emitted, such text SHOULD be related to the certificate, such as providing a textual representation of key data elements in the certificate.

Subject: CN=Atlantis
Issuer: CN=Atlantis
Validity: from 7/9/2012 3:10:38 AM UTC to 7/9/2013 3:10:37 AM UTC

-----BEGIN CERTIFICATE-----
MIIBmTCCAUegAwIBAgIBKjAJBgUrDgMCHQUAMBMxETAPBgNVBAMTCEF0bGFudGlz
MB4XDTEyMDcwOTAzMTAzOFoXDTEzMDcwOTAzM1owEzERMA8GA1UEAxMIQXRs
YW50aXMwXDANBgkqhkiG9w0BAQEFAANLADBIAkEAMBIoAwIBAgIBBwIBAjANBg
ycqzNhRyn/XtkJIIHvYtHvIX+S1x5ErgMoHehycpoxbErmVR4GCq1S2diNmrF2CRTdID
AQABo4GJMIGGMAwGA1UdEwEB/wQICAYDVR0QHBoAAAIB3MDggCgAwIBAgIKABEw
-----END CERTIFICATE-----

Figure 7: Certificate Example with Explanatory Text

5.3. File Extension

Although textual encodings of PKIX structures can occur anywhere, many tools are known to offer an option to output this encoding when serializing PKIX structures. To promote interoperability and to separate DER encodings from textual encodings, the extension ".crt" SHOULD be used for the textual encoding of a certificate. Implementations should be aware that in spite of this recommendation, many tools still default to encode certificates in this textual encoding with the extension ".cer".

This section does not disturb the official application/pkix-cert registration [RFC2585] in any way (which states that "each '.cer' file contains exactly one certificate, encoded in DER format"), but merely articulates a widespread, de facto alternative.
6. Textual Encoding of Certificate Revocation Lists

Certificate Revocation Lists (CRLs) are encoded using the "X509 CRL" label. The encoded data MUST be a BER (DER strongly preferred; see Appendix B) encoded ASN.1 CertificateList structure as described in Section 5 of [RFC5280].

```
-----BEGIN X509 CRL-----
MIIB9DCCAV8CAQEwCwYJKoZIhvcNAQEFMIIBCDEXMBUGAggMVA3mp2Z24sIEluYy4xHzAdBgNVBAsTF12lcmlTaWduIFRydXNoZXI5Q15ldHdvcmsxRjBEBgNVBAsTF12lcmlTaWduIFRydXN0IE5ldHdvcmsxRjBEBgNVBAsTF12lcmlTaWduIFRydXN0IE5ldHdvcmsxRjBEBgNVBAsTF12lcmlTaWduIFRydXN0IE5ldHdvcmsxRjBEBgNVBAsTF12lcmlTaWduIFRydXN0IE5ldHdvcmsxRjBEBgNVBAsTF12lcmlTaWduIFRydXN0IE5ldHdvcmsxRjBE
-----END X509 CRL-----
```

Figure 8: CRL Example

Historically, the label "CRL" has rarely been used. Today, it is not common and many popular tools do not understand the label. Therefore, this document standardizes "X509 CRL" in order to promote interoperability and backwards-compatibility. Generators conforming to this document MUST generate "X509 CRL" labels and MUST NOT generate "CRL" labels. Parsers SHOULD NOT treat "CRL" as equivalent to "X509 CRL". 
7. Textual Encoding of PKCS #10 Certification Request Syntax

PKCS #10 Certification Requests are encoded using the "CERTIFICATE REQUEST" label. The encoded data MUST be a BER (DER strongly preferred; see Appendix B) encoded ASN.1 CertificationRequest structure as described in [RFC2986].

-----BEGIN CERTIFICATE REQUEST-----
MIIBWDCCAQcCAQAwTjELMAkGA1UEBhMCU0UxJzAlBgNVBAoTHlNpbW9uIEpvcz2m
c3NvbIBEYXRhZ29uc3VsdCBQQjEWEQMAAAMBAoJEYkmGHzXUyiyBJhHkPHVzYi
ByqGSM49AgEBSUABHv527b24ub3JnMwGA1uDeWYVDr0PAQH/kBAUADwqADWzgNVH
SUEHEQYMQg1qb3NlZnNzb24ub3JnMwGAYIKwYBBQUAghIIyYBhkOgAwgDAMBOg
-----END CERTIFICATE REQUEST-----

Figure 9: PKCS #10 Example

The label "NEW CERTIFICATE REQUEST" is also in wide use. Generators conforming to this document MUST generate "CERTIFICATE REQUEST" labels. Parsers MAY treat "NEW CERTIFICATE REQUEST" as equivalent to "CERTIFICATE REQUEST".

8. Textual Encoding of PKCS #7 Cryptographic Message Syntax

PKCS #7 Cryptographic Message Syntax structures are encoded using the "PKCS7" label. The encoded data MUST be a BER-encoded ASN.1 ContentInfo structure as described in [RFC2315].

-----BEGIN PKCS7-----
MIHjBgsqhkiG9w0BCRAABgIwJjEwMjB0dBwUExnFzAYIKwYBBQUHAkFIEwYDQY
-----END PKCS7-----

The label "CERTIFICATE CHAIN" has been in use to denote a degenerate PKCS #7 structure that contains only a list of certificates (see Section 9 of [RFC2315]). Several modern tools do not support this label. Generators MUST NOT generate the "CERTIFICATE CHAIN" label. Parsers SHOULD NOT treat "CERTIFICATE CHAIN" as equivalent to "PKCS7".

Josefsson & Leonard Standards Track
PKCS #7 is an old specification that has long been superseded by CMS [RFC5652]. Implementations SHOULD NOT generate PKCS #7 when CMS is an alternative.

9. Textual Encoding of Cryptographic Message Syntax

Cryptographic Message Syntax structures are encoded using the "CMS" label. The encoded data MUST be a BER-encoded ASN.1 ContentInfo structure as described in [RFC5652].

-----BEGIN CMS-----
MIGDBgshkiG9w0BCRABCaB0MHICAQAwDQYLKoZIHvCNAAQQAwwXgYJKoZIhvcN
AQcBoFEEt3icc87PK0nNK9ENqSiTvIOsioo0S/ISczMs1ZIzkgsKk4tsQ0N1nUM
dvb05OXi5XLPETVIvwvLVLwSE0sK1FIVHAqSk3MBkkBAJv0Fx0=
-----END CMS-----

Figure 11: CMS Example

CMS is the IETF successor to PKCS #7. Section 1.1.1 of [RFC5652] describes the changes since PKCS #7 v1.5. Implementations SHOULD generate CMS when it is an alternative, promoting interoperability and forwards-compatibility.

10. One Asymmetric Key and the Textual Encoding of PKCS #8 Private Key Info

Unencrypted PKCS #8 Private Key Information Syntax structures (PrivateKeyInfo), renamed to Asymmetric Key Packages (OneAsymmetricKey), are encoded using the "PRIVATE KEY" label. The encoded data MUST be a BER (DER preferred; see Appendix B) encoded ASN.1 PrivateKeyInfo structure as described in PKCS #8 [RFC5208], or a OneAsymmetricKey structure as described in [RFC5958]. The two are semantically identical and can be distinguished by version number.

-----BEGIN PRIVATE KEY-----
MIGEAgEAMBAGBygSM49AgEGBSuBBAAKBG0wawIBAQQQgVcB/UNPxa1R9zDYAqQIf
jojUD1qGnSJRfEEzZ2T/92hRANCAASc7UJtgnF/abqWM60T3XNJEzBv5ez9TdwK
HQM6xPm2q+53wmsN/eYLdgtjgBd3DMmHtPiCkiFICXyaA8z9LkJ
-----END PRIVATE KEY-----

Figure 12: PKCS #8 PrivateKeyInfo (OneAsymmetricKey) Example
11. Textual Encoding of PKCS #8 Encrypted Private Key Info

Encrypted PKCS #8 Private Key Information Syntax structures (EncryptedPrivateKeyInfo), called the same in [RFC5958], are encoded using the "ENCRYPTED PRIVATE KEY" label. The encoded data MUST be a BER (DER preferred; see Appendix B) encoded ASN.1 EncryptedPrivateKeyInfo structure as described in PKCS #8 [RFC5208] and [RFC5958].

-----BEGIN ENCRYPTED PRIVATE KEY-----
MIHNMEAGCSqGSIb3DQEFDATAzMBsGCSqGSIb3DQEFO3DFAzMBsGCSqGSIb3DQEFO6FzMBsGCSqGSIb3DQEFDAKzMBsGCSqGSIb3DQEFDBABojICA5T/51QICCAAwFAyIkoZIhcCNACRgdg9sB2x dissenting regarding the public key

-----END ENCRYPTED PRIVATE KEY-----

Figure 13: PKCS #8 EncryptedPrivateKeyInfo Example

12. Textual Encoding of Attribute Certificates

Attribute certificates are encoded using the "ATTRIBUTE CERTIFICATE" label. The encoded data MUST be a BER (DER strongly preferred; see Appendix B) encoded ASN.1 AttributeCertificate structure as described in [RFC5755].

-----BEGIN ATTRIBUTE CERTIFICATE-----
MIICKzCCAZQCAQEwFzegZQwYmgYMGcZAJBgNVBAwTAMTREwDwYDVQQI
DAhOZXcgWW9yazEUMBIGA1UEBwwLU3RvbnkgQnJvb2tsbDANBgNVBAoMBkNTRTU5
MjE6MDgGA1UEAxwUZ2NvdHJvbGxlc3I9I1bWFpDQFkZ21cMnRc3N0YWxsZXMj
aWMuc3VuteXNlUmVkdQFugQyGgIGMMIGJpIGGMIGDMQswCQYDVQQGEwJVUTER
MA8GA1UEAwJqMzIvSm9qZmw0LDEwFQYDVQQKEwJqMzIvMjAxMzIvMTIwMjEwPjA6
DQYJKoZIhvcNAQEFQACBMAgEAMG0GMA8GA1UEChMhITEyMzAxMzIvMTIwMjEwPjA6
DQYJKoZIhvcNAQEFQACBAAMBEWcCBKUwDQYJKoZIhvcNAQEFQACBAAMBEWcCBKUw
-----END ATTRIBUTE CERTIFICATE-----

Figure 14: Attribute Certificate Example
13. Textual Encoding of Subject Public Key Info

Public keys are encoded using the "PUBLIC KEY" label. The encoded data MUST be a BER (DER preferred; see Appendix B) encoded ASN.1 SubjectPublicKeyInfo structure as described in Section 4.1.2.7 of [RFC5280].

-----BEGIN PUBLIC KEY-----
MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEn1LlwLN/KBYQRVH6HfINTzfeqJ0VztLe
kLchp2h178cCaMY81FB1ys8J917krc+M4aBeCGYFjba+hiXttJWPL7yd1E+5UG4U
Nkn3Eos8ElZByi9DVsyfy9eejh+8AXgp
-----END PUBLIC KEY-----

Figure 15: Subject Public Key Info Example

14. Security Considerations

Data in this format often originates from untrusted sources, thus parsers must be prepared to handle unexpected data without causing security vulnerabilities.

Implementers building implementations that rely on canonical representation or the ability to fingerprint a particular data object need to understand that this document does not define canonical encodings. The first ambiguity is introduced by permitting the text-encoded representation instead of the binary BER or DER encodings, but further ambiguities arise when multiple labels are treated as similar. Variations of whitespace and non-base64 alphabetic characters can create further ambiguities. Data encoding ambiguities also create opportunities for side channels. If canonical encodings are desired, the encoded structure must be decoded and processed into a canonical form (namely, DER encoding).

15. References

15.1. Normative References


Informative References


RFC 5208, May 2008, 


Appendix A. Non-conforming Examples

This appendix contains examples for the non-recommended label variants described earlier in this document. As discussed earlier, supporting these is not required and is sometimes discouraged. Still, they can be useful for interoperability testing and for easy reference.

-----BEGIN X509 CERTIFICATE-----
MIIBHDCBxaADAgECAgIcxzAJBgcqqhkjOPQQBMBAxjAMBgNVBAMUBVBLSVghMB4X
DTEOMdxxNDA2MTU1MCoXDTI0MDxxNDA2MTU1MFOwEDEOMAwGA1UEAxQFUEtJWCEw
WTATBgchhkjOPQIBBqqhkjoFQMBBwNCATwoQSR863Qr0PoRIYQ96H7WyKePH
WaoeVAE24bth43wCNC+U5aZ761dGhSSJKvVXRgVH5+prlIrznf1iq+X4oxAwDjAM
BgNHVRMBAf8E8JAAAMAKGByqGSM49BAEDRwAwRAIfMdKS5F631MnWWh7uaKJzKCs
NnY/OKgBex6MIEAv2AIhAI2GdvF+L+mGvhyPXE+JxRxxWChmggb5/9eHdUcmW/jkOH
-----END X509 CERTIFICATE-----

Figure 16: Non-standard 'X509' Certificate Example

-----BEGIN X.509 CERTIFICATE-----
MIIBHDCBxaADAgECAgIcxzAJBgcqqhkjOPQQBMBAxjAMBgNVBAMUBVBLSVghMB4X
DTEOMdxxNDA2MTU1MCoXDTI0MDxxNDA2MTU1MFOwEDEOMAwGA1UEAxQFUEtJWCEw
WTATBgchhkjoFQMBBwNCATwoQSR863Qr0PoRIYQ96H7WyKePH
WaoeVAE24bth43wCNC+U5aZ761dGhSSJKvVXRgVH5+prlIrznf1iq+X4oxAwDjAM
BgNHVRMBAf8E8JAAAMAKGByqGSM49BAEDRwAwRAIfMdKS5F631MnWWh7uaKJzKCs
NnY/OKgBex6MIEAv2AIhAI2GdvF+L+mGvhyPXE+JxRxxWChmggb5/9eHdUcmW/jkOH
-----END X.509 CERTIFICATE-----

Figure 17: Non-standard 'X.509' Certificate Example

-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBWDCCAcCAIAAwTjELMAKGA1UEBhMCUGFUaTh1NpbW9uIEpvc2Vz
C3NvbIBhXcaEBq5/9CeBGABQGA1UEAxMNam9zZjZ2s29ulm9yZzZOMBAG
ByqGSM49AgEBBubeAAhAzoABLPLPSkuX016MboV3J3ot5FCuqfn6dTs+9/CM
EO1SwVej7t7j5k9j9q/LfysX8FO915p3oGIwYAYJKoZIhvcNAQkOMVwUTAY
BgNVHREEEAPgg1qb3N12nNzb24ub3JnMwGA1UdeEB/wQCMAAwDwYDVQR0PAQH/
BAUDAwEcADABWQgNVHSUBAf8EDDABKgrgBFQcDQAKBggqhkjoFQDDAgm/ADA8
AHzvBvhirPPFpFbBbsE1NoFmCuxOFAEuQVUw32P69AhwWXk3dGUSKnuwL5g/ftAY
dEqc8BjAcunu0rOfu
-----END NEW CERTIFICATE REQUEST-----

Figure 18: Non-standard 'NEW' PKCS #10 Example
-----BEGIN CERTIFICATE CHAIN-----
MIHjBgsqhk1G9w0BcrABC6C05G0AIBADFH018CAQCgWYJKoZIhvGAQUMMA4E
ClfrI6dr0gUMAgIT1dDAjBgsqhk1G9w0BcRAACUAABgqWhk1G9w0DBwPOLZpEERh
u5kEGDCjBerXYodQ7EErUZjvAurk/jB1IrozBSBkgqWhk1G9w0BBwEwLYKoZl
hvcNAQkQAw8WJDADUBgqWhk1G9w0DBwQ1QcCBrU9nxEwDAIJKwYBBQUIAQQIFAI
AQQ0sYGYUfDA0RqN1p4VbKEAQUM2Xo8PMHBoYdqEcsbTodlCFAZH4=
-----END CERTIFICATE CHAIN-----

Figure 19: Non-standard 'CERTIFICATE CHAIN' Example

Appendix B. DER Expectations

This appendix is informative. Consult the respective standards for
the normative rules.

DER is a restricted profile of BER [X.690]; thus, all DER encodings
of data values are BER encodings, but just one of the BER encodings
is the DER encoding for a data value. Canonical encoding matters
when performing cryptographic operations; additionally, canonical
encoding has certain efficiency advantages for parsers. There are
three principal reasons to encode with DER:

1. A digital signature is (supposed to be) computed over the DER
encoding of the semantic content, so providing anything other
than the DER encoding is senseless. (In practice, an implementer
might choose to have an implementation parse and digest the data
as is, but this practice amounts to guesswork.)

2. In practice, cryptographic hashes are computed over the DER
encoding for identification.

3. In practice, the content is small. DER always encodes data
values in definite-length form (where the length is stated at the
beginning of the encoding); thus, a parser can anticipate memory
or resource usage up front.
Figure 20 matches the structures in this document with the particular reasons for DER encoding:

<table>
<thead>
<tr>
<th>Sec.</th>
<th>Label</th>
<th>Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>CERTIFICATE</td>
<td>1 2 ~3</td>
</tr>
<tr>
<td>6</td>
<td>X509 CRL</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>CERTIFICATE REQUEST</td>
<td>1 ~3</td>
</tr>
<tr>
<td>8</td>
<td>PKCS7</td>
<td>*</td>
</tr>
<tr>
<td>9</td>
<td>CMS</td>
<td>*</td>
</tr>
<tr>
<td>10</td>
<td>PRIVATE KEY</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>ENCRYPTED PRIVATE KEY</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>ATTRIBUTE CERTIFICATE</td>
<td>1 ~3</td>
</tr>
<tr>
<td>13</td>
<td>PUBLIC KEY</td>
<td>2 3</td>
</tr>
</tbody>
</table>

Figure 20: Guide for DER Encoding

* Cryptographic Message Syntax is designed for content of any length; indefinite-length encoding enables one-pass processing (streaming) when generating the encoding. Only certain parts -- namely, signed and authenticated attributes -- need to be DER encoded.

- Although not always "small", these encoded structures should not be particularly "large" (e.g., more than 16 kilobytes). The parser ought to be informed of large things up front in any event; this is yet another reason to DER encode these things in the first place.

Figure 20: Guide for DER Encoding

Acknowledgements

Peter Gutmann suggested to document labels for Attribute Certificates and PKCS #7 messages, and to add examples for the non-standard variants. Dr. Stephen Henson suggested distinguishing when BER versus DER is appropriate or necessary.
Authors’ Addresses

Simon Josefsson
SJD AB
Johan Olof Wallins Vaeg 13
Solna 171 64
Sweden

EMail: simon@josefsson.org
URI: http://josefsson.org/

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

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