Textual Encodings of PKIX, PKCS, and CMS Structures

draft-josefsson-pkix-textual-09

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 is intended to articulate the de-facto rules that existing implementations operate by, and to give recommendations that will promote interoperability.

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 June 16, 2015.

Copyright Notice

Copyright (c) 2014 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 standards.
6. Attribute Certificates in An Internet Attribute Certificate Profile for Authorization
Although other formats exist that use the encodings (or something like them) described in this document, the included formats share a common property: algorithm agility. "Algorithm agility" means that different algorithms to achieve the same purposes—such as content encryption or integrity protection—can be used in different instances of the same format because the instance data identifies the algorithms and associated parameters. Weakness in an algorithm does not destroy the utility of the format.

A disadvantage of a binary data format is that it cannot be interchanged in textual transports, such as e-mail 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 PEM [RFC1421], based on a proposal by M. Rose in Message Encapsulation [RFC0934]. 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 [RFC2015] 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.

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 starting with `-----BEGIN` and ends with a line starting with `-----END`. Between these lines, or "encapsulation boundaries", are base64-encoded data according to Section 4 of [RFC4648]. Data before the `-----BEGIN` and after the `-----END` encapsulation boundaries are permitted and parsers MUST NOT malfunction when processing such data. Furthermore, parsers MUST 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. Parsers MAY disregard the label on the `-----END` line instead of signaling an error if there is a label mismatch. There is exactly one space character (\SP\) separating the BEGIN or END from the label. There are exactly five hyphen-minus (or 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 Internet-Draft 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. Consistent with algorithm agility, the labels described in this document are not specific to any particular cryptographic algorithm.

Unlike legacy PEM encoding [RFC1421], OpenPGP ASCII armor, and the OpenSSH key file format, textual encoding does not define or permit attributes to be encoded alongside the PKIX or CMS data. Whitespace MAY appear between the pre-encapsulation boundary and the base64, but generators SHOULD NOT emit such whitespace.

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.

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). Parsers MAY handle other line sizes. These requirements are consistent with PEM [RFC1421].

3. ABNF

The ABNF [RFC5234] of the textual encoding is:

```
pkixmsg   = preeb
  *eolWSP
  base64text
  posteb

preeb     = "-----BEGIN " label "-----" eol

posteb    = "-----END " label "-----" eol

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

base64pad = "="

base64line = 1*base64char eol

base64finl = *base64char (base64pad eol base64pad / *2base64pad) eol
  ; ...AB= <CRLF> = <CRLF> 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
```
New implementations SHOULD emit the strict format [abnf-strict-fig] specified above.

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>[RFC5958] 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 3: Convenience Guide
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) encoded ASN.1 Certificate structure as described in section 4 of [RFC5280].

```
-----BEGIN CERTIFICATE-----
MIICLDCCAdKgAwIBAgIBADAKBggqhkjOPQDQJ9MQswCQYDVQQGEwJCRTEPMA0GA1UE
ChMGR251VExTMSUwIwYDVQQLExxHbnWnhdGugYXV0aG9ya251XzIwMTAwMVYDVQQJe
FJc5XzIwMTAwMVYDVQQLExxHbnWnhdGugYXV0aG9ya251XzIwMTAwMVYDVQQJeFJc
-----END CERTIFICATE-----
```

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 are NOT RECOMMENDED to 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-----
MIIBmTCACUAgAwIBAgIBKjAJBgUrDgMCHGUAMBMxETAPBgNVBAMTCEF0bGFudGlz
MB4XDTExMDc0OTAzMDIzMDEwMDAwMDAwMDAwMDBUMQwwCQgMAwGA1UdEwEB/wQFAw
-----END CERTIFICATE-----
```
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.

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) encoded ASN.1 CertificateList structure as described in Section 5 of [RFC5280].

```
-----BEGIN X509 CRL-----
MIIB9DCCAV8CAQEwCwYJKoZIhvcNAQEFMIIBCDEXMBUGA1UEChMOVmVyaVNpZ24s
IEluYy4xHzdAdBgvNBAsTF1Z1cmlTaWduIFRydXN0IE5ldHdvcmsxRjBEbgNVBAstT
PX3d3y5ZfJpc21bi5j5b20vcmVwbn3NpdG9yeS9SUEEgSW5j5b3JwLibieSBszWyYu
LExJQiUtTFREKGMPoQTgxHjaCbgNVBAstTFVBLcnNvbmgEgTm90IFZhbGlkJXRIZDEm
MCQGA1UECxMrGlnaXRhbCBJREBJRCBDBdGFzcyAxICogTmV0c2NhCGQDAWBgNVAMU
D1NpbW9uIEpv2Vmc3NvbjjE1MACGCSqGSIb3DQEJARYTc2ltb25Aaam9zZWZzc29u
Lm9y9xcNMDYyMj13MDgwMjQWWhcNMDcwMjA3MDgwMj1WjA5MjCECECCACQRQW6FrOw
el1UNpllhTgXgTA2MTiyNz4A4MDIzNFDowCwYJKoZIhvcNAQEFA4GBAD0zX+J2khcc
Nbrq1Dn5IKL8nXLgGChv1I/le1MNo9t1ohGQxSBhFkRPAAY82fr6Epor4aHgVy
b+5y+neKNKmPF4iun+a4o26Cj3p0arojCL1p8z0yy9xxvyczsaZ98H1lyP
c3DMGR+oUm5jKZ0J1AYmeLxaPHFQwR
-----END X509 CRL-----
```

Figure 7: 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) encoded ASN.1 CertificationRequest structure as described in [RFC2986].

```
-----BEGIN CERTIFICATE REQUEST-----
MIIBWDCCAQcCwOAwTjELMakaGIAUEBhMCU0UXJzALBgNVBAOTh1NpbW9uIIEpvc2Vm
c3NvbiBEYXRh29uc3VsdCBQQEJEMBQGA1UEAxMNaN9zZWZzc29uLm9yZzBOMBAG
-----END CERTIFICATE REQUEST-----
```

Figure 6: Certificate Example with Explanatory Text
Figure 8: 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-----
MIHjBgsqhkiG9w0BCRBgZCSRATCBzAIAFkDho18CAQCGbIK0ZlY3RvY3AsCQx9
MAcGCSqGSIb3DQEBCwUAA4GQCbcgMl7g9Bk859JsMTM9yHwCgYEA397H81lyx
JHRJ1d712wFm92b72Vz0uPnMf0zBk712jJGgY2J01Zb
-----END PKCS7-----
```

Figure 9: PKCS #7 Example

The label `CERTIFICATE CHAIN` has been in use to denote a degenerative PKCS #7 structure that contains only a list of certificates. 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`.

PKCS #7 is an old standard 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-----
MIGDBgsqhkiG9w0BCRBgZCSRATCBzAIAFkDho18CAQCGbIK0ZlY3RvY3AsCQx9
MAcGCSqGSIb3DQEBCwUAA4GQCbcgMl7g9Bk859JsMTM9yHwCgYEA397H81lyx
JHRJ1d712wFm92b72Vz0uPnMf0zBk712jJGgY2J01Zb
-----END CMS-----
```

Figure 10: 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. Textual Encoding of PKCS #8 Private Key Info, and One Asymmetric Key

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) 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-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgCgIj85oX3B/2B46G3c35zu+i5W
AcJ+o2/4WcV1B4wwcAgyTv+Q9zq7FopB95Q/1ud/8H6z5ye4567X6L3H0hZ59V
-----END PRIVATE KEY-----
```

Figure 11: 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) encoded ASN.1 EncryptedPrivateKeyInfo structure as described in PKCS #8 [RFC5208] and [RFC5958].

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIHNMEAGCSqGSIb3DQEFDTAzMBsGCSqGSIb3DQFEFDDAOBAgghhICA6T/5IQCICAAs
-----END ENCRYPTED PRIVATE KEY-----
```

Figure 12: 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) encoded ASN.1 AttributeCertificate structure as described in [RFC5755].

```
-----BEGIN ATTRIBUTE CERTIFICATE-----
MIICZzCCAZQCAQEvwCzggZQwZmkBYwYQyCzAzAJBgNVBAYTAlVTMREwDwYDVQQI
DAhOZXcgWW9yazEUMBIGA1UEBwwLU3RvbnkgQnJvb2sxFDASBgNVBAoMMVNjb3R0IFN0YWxsZXIvZ
-----END ATTRIBUTE CERTIFICATE-----
```

9 of 13
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) encoded ASN.1 `SubjectPublicKeyInfo` structure as described in Section 4.1.2.7 of [RFC5280].

```
-----BEGIN PUBLIC KEY-----
MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEn1LlwLN/KBYQRVH6HFIMTzfEqJ0VztLe
kLchp2hi78cCaMY81FB1ys8J917krc+M4aBeCGYFjba+hiXttJWPL7ydeE+5UG4U
Nkn3Eos8EiZbyi9DVsyfy9eejh+8AXgp
-----END PUBLIC KEY-----
```

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 Internet-Draft 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. IANA Considerations

This document implies no IANA Considerations.

16. 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 are appropriate or necessary.

17. References

17.1. Normative References

Informative References


Appendix A. Non-Conforming Examples

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

<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 19 matches the structures in this document with the particular reasons 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, which is yet another reason to DER encode these things in the first place.

Figure 19: Guide for DER Encoding

Authors’ Addresses

**Simon Josefsson**
SJD AB
Johan Olof Wallins Våg 13
Solna, 171 64
SE
EMail: simon@josefsson.org
URL: http://josefsson.org/

**Sean Leonard**
Penango, Inc.
5900 Wilshire Boulevard
21st Floor
Los Angeles, CA 90036
USA
EMail: dev+ietf@seantek.com
URL: http://www.penango.com/