Textual Encodings of PKIX, PKCS, and CMS Structures
draft-josefsson-pkix-textual-10

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, defines them so that future implementations can interoperate.

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 July 2, 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.
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 Marshall 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. 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 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 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. 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 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. Label</th>
<th>ASN.1 Type</th>
<th>Reference Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 CERTIFICATE</td>
<td>Certificate</td>
<td>[RFC5280] id-pkix1-e</td>
</tr>
<tr>
<td>6 X509 CRL</td>
<td>CertificateList</td>
<td>[RFC5280] id-pkix1-e</td>
</tr>
<tr>
<td>7 CERTIFICATE REQUEST</td>
<td>CertificationRequest</td>
<td>[RFC2986] id-pkcs10</td>
</tr>
<tr>
<td>8 PKCS7</td>
<td>ContentInfo</td>
<td>[RFC2315] id-pkcs7*</td>
</tr>
<tr>
<td>9 CMS</td>
<td>ContentInfo</td>
<td>[RFC5652] id-cms2004</td>
</tr>
<tr>
<td>10 PRIVATE KEY</td>
<td>PrivateKeyInfo ::=</td>
<td>[RFC5208] id-pkcs8</td>
</tr>
<tr>
<td>11 ENCRYPTED PRIVATE KEY</td>
<td>EncryptedPrivateKeyInfo</td>
<td>[RFC5958] id-aKPV1</td>
</tr>
<tr>
<td>12 ATTRIBUTE CERTIFICATE</td>
<td>AttributeCertificate</td>
<td>[RFC5755] id-acv2</td>
</tr>
<tr>
<td>13 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].
Historically the label \texttt{X509 CERTIFICATE} and also less commonly \texttt{X.509 CERTIFICATE} have been used. Generators conforming to this document MUST generate \texttt{CERTIFICATE} labels and MUST NOT generate \texttt{X509 CERTIFICATE} or \texttt{X.509 CERTIFICATE} labels. Parsers SHOULD NOT treat \texttt{X509 CERTIFICATE} or \texttt{X.509 CERTIFICATE} as equivalent to \texttt{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-----
MIICDCCAdKgAwIBAgIBDADAKBggqhjkjOPQDAjB9MQswCQYDVQQGEwJCRTEMA0GAIUEChMRG2S1VEtMUSUwIwYDVQQLEExHbnVUTFMsgY2YdG1maWNhdGUXV9YXV0aG9y
aXR5MQ8wDQYDVQQIEwZMXVZ2Z4xJTAgBgNVBAMTExdudVRMuyEyBJZj0aWZpY2F0
ZSBhdXR0b3JpdHkwHhcNMTEwNTIzMjIzODIxWhcNMTIzMjIyMDC0MTUxWjB9MQsw
CQYDVQQGEwJCRTEMA0GAIUEChMRG2S1VEtMUSUwIwYDVQQLEExHbnVUTFMsgY2YdG1maWNhdGUXV9YXV0aG9yaXR5MQ8wDQYDVQQIEwZMXVZ2Z4xJTAgBgNBAMTExdud
VRMuyEyBJZj0aWZpY2F0ZSBhdXR0b3JpdHkwWTATBgqc中国opk0PQIBBgghkj0PQMB
BuNCARRS2I0jiUnYi14UzSALCX3iybqijIUvxpj+jONn8ngv/j/Niyvz394BWnW4x
uxQ4RTiyxkw87wrwMgJG6Z5k/72no0mQTPBhgVNRMFABfE8ETDADQH/MA8GA1ud
DweB/wQFwMVMBGAgwQVwQR0BOBwFEC0g6F6YEr+1KLkQAPLzB9mTigDMwCcqG
SM49BAMCAA8gAMEUCIDGwuD1KPyg+hRf88MeyMQcQFZDT0TBVleF+UsAGQ4eAIb
l4WoUdkQ+aupc8GftXE2C//4mKANBC6I0lT1po=
-----END CERTIFICATE-----

Figure 6: Certificate Example

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 \texttt{.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 \texttt{.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 \texttt{X509 CRL} label. The encoded data MUST be a BER (DER strongly preferred; see Appendix B) encoded ASN.1 \texttt{CertificateList} structure as described in Section 5 of [RFC5280].

\begin{verbatim}
-----BEGIN X509 CRL-----
MIIBB9DCCAV8CAQjEwYJIoZIhvcNAQEUMBIGCDExMBUGA1UEChM0MvVyVNpZ24s
IEluYy4xHzAdBgNVAsTFzlcmTawduIFRydXN0IE5ldHdvcmxRjBEbgNVBAst
PXd3dy52ZXJpc2lnbi5jb20vcMvVw3NpdG9yeS9SUEgSWj3JWLibieSB5ZWYu
LEkJQuiTFREKGMpOTgxJjAcBgNVBAsgVFBlcnVbemEgTm90IFzhbGlkYXRlZDEm
MCQGAvUCAxXEdG1nXRhhCJRCBDBDGFzyAxCigV0cZ2NhcgGUxGDAWBgNVBAoMD
1NPbW9uIEpvc2Vmc3NvbjEiMCAGCSqGsG1b3DQEJARYTc2ltb25Aam9zWZzc29u
Lm9yZxcNMdYxMjI3MDgwMjM0WbcNMdcwMjA3MDgwMjM1WjA1MCECC4QNPwRoWd
eUNplhhbNgDXTA2MTiyNzA4MDIzNfOWCwYJXoZihvcNAQEFA4GBADQ0Z+J2hke
Nbrq1dn5ILKnXLggPCHvul/I/le1MKn09t10gHQxB5HnFuRPA82r6Eor4AhGvy
b+5y+eneK9NKn2mpF4iium+a4o26Cj1opArojCL1p8T0yyi9Xyvyc/ezA98HiLyP
c3DMGMR+oUmSjKZ0jIhAYmeLxaPHFqW
-----END X509 CRL-----
\end{verbatim}

Figure 8: CRL Example

Historically the label \texttt{CRL} has rarely been used. Today it is not common and many popular tools do not understand the label. Therefore, this document standardizes \texttt{X509 CRL} in order to promote interoperability and backwards-compatibility. Generators conforming to this document MUST generate \texttt{X509 CRL} labels and MUST NOT generate \texttt{CRL} labels. Parsers SHOULD NOT treat \texttt{CRL} as equivalent to \texttt{X509 CRL}.

7. Textual Encoding of PKCS #10 Certification Request Syntax

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

\begin{verbatim}
-----BEGIN CERTIFICATE REQUEST-----
MIIBWDCCAQCCAQjWTeJLMawGA1UEBhMCU0UxJzAlBQgNVBAoTH1NpbW9uIEpvc2Vmc3NvbiBEYXRhak29uc3VsdCBQQjEwMBQGA1UEAxMNam9zWZzc29uLm9yZzBMA8
ByqGSM49AgEBGUuBBABAhAzoABLLPSkuXy0166MBvXJ3Mot5FCFuuqQmn6Ds+9/CME0lsWJe77tj56k9JR+jQ+LfySx8FO15p3oGiwYAYJKoZihvcNAQkOMMwUTAY
BgNVREETAPggqbp3N1ZnZnb24ub32nMwGA1udEwEB/wQCMAAwDwYVDR0PAQH/BAUDAwegADAB8NgHVSUBAaF8EDDAK8gBgEFBQcDATAKBggnkjq0PQODM/A/AD8A
Dq2BvHmXPfBmBbEs1OfMxCuczOFAPEuQVUw3ZP69AhwWXk3dgSUuKL5/g/feAY
-----END CERTIFICATE REQUEST-----
\end{verbatim}

Figure 9: PKCS #10 Example

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

Josefsson & Leonard
Expires July 2, 2015
Page 8 of 14
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-----
MIHjBgsqhhkiG9w0BCRABF6CB0zCB0AIBADFho18CAQCgWyJKoZIhvcNAQUMMA4E
CIfrI6dr0gUNAgITiDAjBgsqhhkiG9w0BCRADCTAUBggqhkiG9w0DBwQIZpECRWtz
u5kEGDcjerXY8odQ7EEeromZjvAurk/j81IrozBSBqkhkiG9w0BBwEwYWLYKoZI
hvcNAQkQAw8wJDAUBggqhkiG9w0DBwQI0tCCu09nxEwDAYIKwYBBQUIAQIFAIQAQ
0sYQYUFdAH0Rnc1p4VbKEAQUM2Xo8PMHB0ydqEcsbTodlCFAZH4=
-----END PKCS7-----
```

Figure 10: PKCS #7 Example

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.

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-----
MIgDBgsqhhkiG9w0BSCRABCaB0MHICAQAwDQLKoZIhvcNAQkQAwgXgYJKoZIhvcN
AQcBoFETT3icc87PK0nNK9ENqSxItVLoSao0o0S/I5czMs1ZIzkgsKk4tsQ01nUM
dvb050X15XLPLEtViMwvLw5E0sK1FIVHAqS5k3MBkkBAJu0Fx0=
-----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. 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; 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.
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-----
MIHNMAGCSqGS1b3DQFEDTAzMBsGCSqGS1b3DQFEFDAOBAGhhICA6T/51QICCAAw
FAYIKoZIhvcNAwcECBCxvDg5f159j3BIGIY3CAq1MNBoaS1iqiWVNJ3Ipfnbiefw
Z0jIIOyRmk/K/+crq90PlnzmImmOTR9s4jrG3cilzTWvb0jIvbG3hu0zyFPraoMkap
8erZwsIc5SVel+CSjoSz2mVS87cy1Jd+txrMrXOVYDE+eTGMLbrLmsWh3QKCrtF
QC7k0NnzUHTV9yGDwqfMbw==
-----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-----
MIICKzCZACQCAEQwZeggZQwYmkgYYmgYmXczAJBgNVBAYTAlVTMReDwYDVQQI
DAhOZCXcgW9yazEUMBIGA1UEBwvuU3Rvbngq0njb2xszDzANBgNVBAoMBKNTTU5
MJj6MDaGA1UEAxwwU2Nvhq03RhbGxcl19ibWFpbEFkZHJlc3M9c3N0YXRlZXJxa
wMuc3VxeX1lmVkdqWHaQnGSMlGiqIGGMIQMDQswCQYDVQQGEwJvUZER
MA8GA1UECamwLTmV3IFlvcmsxFDA8BgNBAdMC0uN0b255IEyb29rMQ8wQDYDVQQK
DAZDU0U01TIXojA4BqNVBAMMMVnb3R0IFN0YWxszX1sZ1haWxSBG RyZXNzP2x
6GFsbGVyQG1jLzNnbnLzYi51ZHUwDQYJKoZIhvcNAQEFBQCGBQVf4FSjAigAz
OTA3M1d1wMTA1MDAwMwYzMSMTEwMTMxMjUwMDAwJwArMCKGA1UESdEiCCMCGhm
rDHA6Ly9pZG4yXNOi5vcmcvawW5kZXguARtbDANBgkqkhiG9w0BAQFDAOBgAA
M9aXFPXozEFcer06bj9MCBBCQltaA7ZXCijcyjva7xCBdmtZXPYUuHF5OcWPJz
5XPus/sX9wBGtLMDflIKNyX08Rsm60cx+PGUICc7zp2GmCYLl641AEGP0/bsw
SmakuA1ZlIttePETAHeJJs8izJN5j43WcdA5gLztQ==
-----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].
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

17.2. 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.

```
-----BEGIN X509 CERTIFICATE-----
MIIBHDCAxADAECAGICxZAJBgqhkjOPQ8MBAxDjAMBgNVBAAMBVBLSVghMB4X
    DTE0MDkxNDA2MTU1MFowEDEOMAwGA1UEAxQFUEEtJWCEw
WTATBgqhkjOPQIBBggghkjOPQMBBwNCAAAtwoQSi863Qr0PoRIYQ96H7WykDePH
    Wa0eVAE24bth43wCNc+U5azZ761dhGhSSJkVWRgVH5+prLIR+nzfIq+X4oxAwDjAM
    BgNVHRMA8EjA8MAkGBqyGSM49BAEDRwAwRAIfMdKS5F631MnWVi7uaKJzKCs
    NnY/OKgBexMIEAv2AlhAI2GdvFL+mGvhyPZE+JxRxWChmggb5/9eHdUcmW/jkOH
-----END X509 CERTIFICATE-----
```

Figure 16: Non-standard 'X509' Certificate 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:
*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 20: Guide for DER Encoding**

**Authors’ Addresses**

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

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