LAMPS
Internet-Draft
Updates: 5280 (if approved)
Intended status: Standards Track
Expires: September 5, 2018

A. Melnikov, Ed.
Isode Ltd
W. Chuang, Ed.
Google, Inc.
March 4, 2018

Internationalized Email Addresses in X.509 certificates
draft-ietf-lamps-eai-addresses-18

Abstract

This document defines a new name form for inclusion in the otherName field of an X.509 Subject Alternative Name and Issuer Alternative Name extension that allows a certificate subject to be associated with an Internationalized Email Address.

This document updates RFC 5280.

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 https://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 5, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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.

Table of Contents

1. Introduction

[RFC5280] defines the rfc822Name subjectAltName name type for representing [RFC5321] email addresses. The syntax of rfc822Name is restricted to a subset of US-ASCII characters and thus can't be used to represent Internationalized Email addresses [RFC6531]. This document defines a new otherName variant to represent Internationalized Email addresses. In addition this document requires all email address domains in X.509 certificates to conform to IDNA2008 [RFC5890].

2. Conventions Used in This Document

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 [RFC2119].

The formal syntax uses the Augmented Backus-Naur Form (ABNF) [RFC5234] notation.

3. Name Definitions

The GeneralName structure is defined in [RFC5280], and supports many different name forms including otherName for extensibility. This section specifies the SmtpUTF8Mailbox name form of otherName, so that Internationalized Email addresses can appear in the subjectAltName of

a certificate, the issuerAltName of a certificate, or anywhere else that GeneralName is used.

```
id-on-SmtpUTF8Mailbox OBJECT IDENTIFIER ::= { id-on 9 }

SmtpUTF8Mailbox ::= UTF8String (SIZE (1..MAX))
-- SmtpUTF8Mailbox conforms to Mailbox as specified
-- in Section 3.3 of RFC 6531.
```

When the subjectAltName (or issuerAltName) extension contains an Internationalized Email address with a non-ASCII local-part, the address MUST be stored in the SmtpUTF8Mailbox name form of otherName. The format of SmtpUTF8Mailbox is defined as the ABNF rule SmtpUTF8Mailbox. SmtpUTF8Mailbox is a modified version of the Internationalized Mailbox which was defined in Section 3.3 of [RFC6531] which was itself derived from SMTP Mailbox from Section 4.1.2 of [RFC5321]. [RFC6531] defines the following ABNF rules for Mailbox whose parts are modified for internationalization: <Local-part>, <Dot-string>, <Quoted-string>, <QcontentSMTP>, <Domain>, and <Atom>. In particular, <Local-part> was updated to also support UTF8-non-ascii. UTF8-non-ascii was described by Section 3.1 of [RFC6532]. Also, domain was extended to support U-labels, as defined in [RFC5890].

This document further refines Internationalized [RFC6531] Mailbox ABNF rules and calls this SmtpUTF8Mailbox. In SmtpUTF8Mailbox, labels that include non-ASCII characters MUST be stored in U-label (rather than A-label) [RFC5890] form. This restriction removes the need to determine which label encoding A- or U-label is present in the Domain. As per Section 2.3.2.1 of [RFC5890], U-label are encoded as UTF-8 [RFC3629] in Normalization Form C and other properties specified there. In SmtpUTF8Mailbox, domain labels that solely use ASCII characters (meaning not A- nor U-labels) SHALL use NR-LDH restrictions as specified by Section 2.3.1 of [RFC5890] and SHALL be restricted to lower case letters. NR-LDH stands for "Non-Reserved Letters Digits Hyphen" and is the set of LDH labels that do not have "--" characters in the third and forth character position, which excludes "tagged domain names" such as A-labels. Consistent with the treatment of rfc822Name in [RFC5280], SmtpUTF8Mailbox is an envelope <Mailbox> and has no phrase (such as a common name) before it, has no comment (text surrounded in parentheses) after it, and is not surrounded by "<" and ">".

Due to name constraint compatibility reasons described in Section 6, SmtpUTF8Mailbox subjectAltName MUST NOT be used unless the local-part of the email address contains non-ASCII characters. When the local-part is ASCII, rfc822Name subjectAltName MUST be used instead of SmtpUTF8Mailbox. This is compatible with legacy software that

supports only rfc822Name (and not SmtpUTF8Mailbox). The appropriate usage of rfc822Name and SmtpUTF8Mailbox is summarized in Table 1 below.

SmtpUTF8Mailbox is encoded as UTF8String. The UTF8String encoding MUST NOT contain a Byte-Order- Mark (BOM) [RFC3629] to aid consistency across implementations particularly for comparison.

local-part char	domain char	domain label	subjectAltName
ASCII-only	ASCII-only	NR-LDH label	rfc822Name
non-ASCII	ASCII-only	NR-LDH label	SmtpUTF8Mailbox
ASCII-only	non-ASCII	A-label	rfc822Name
non-ASCII	non-ASCII	U-label	SmtpUTF8Mailbox

non-ASCII may additionally include ASCII characters.

Table 1: Email address formatting

4. IDNA2008

To facilitate comparison between email addresses, all email address domains in X.509 certificates MUST conform to IDNA2008 [RFC5890] (and avoid any "mappings" mentioned in that document). Use of non-conforming email address domains introduces the possibility of conversion errors between alternate forms. This applies to SmtpUTF8Mailbox and rfc822Name in subjectAltName, issuerAltName and anywhere else that these are used.

5. Matching of Internationalized Email Addresses in X.509 certificates

In equivalence comparison with SmtpUTF8Mailbox, there may be some setup work on one or both inputs depending of whether the input is already in comparison form. Comparing SmtpUTF8Mailboxs consists of a domain part step and a local-part step. The comparison form for local-parts is always UTF-8. The comparison form for domain parts depends on context. While some contexts such as certificate path validation in [RFC5280] specify transforming domain to A-label (Section 7.5 and 7.2 in [RFC5280] as updated by [ID-lamps-rfc5280-i18n-update]), this document recommends transforming to UTF-8 U-label instead. This reduces the likelihood of errors by reducing conversions as more implementations natively support U-label domains.

Comparison of two SmtpUTF8Mailbox is straightforward with no setup work needed. They are considered equivalent if there is an exact

octet-for-octet match. Comparison with email addresses such as Internationalized email address or rfc822Name requires additional setup steps for domain part and local-part. The initial preparation for the email addresses is to remove any phrases or comments, as well as "<" and ">" present. This document calls for comparison of domain labels that include non-ASCII characters be transformed to U-label if not already in that form. The first step is to detect use of the A-label by using Section 5.1 of [RFC5891]. Next if necessary, transform any A-labels to U-labels Unicode as specified in Section 5.2 of [RFC5891]. Finally if necessary convert the Unicode to UTF-8 as specified in Section 3 of [RFC3629]. For ASCII NR-LDH labels, upper case letters are converted to lower case letters. In setup for SmtpUTF8Mailbox, the email address local-part MUST conform to the requirements of [RFC6530] and [RFC6531], including being a string in UTF-8 form. In particular, the local-part MUST NOT be transformed in any way, such as by doing case folding or normalization of any kind. The <Local-part> part of an Internationalized email address is already in UTF-8. For rfc822Name the local-part, which is IA5String (ASCII), trivially maps to UTF-8 without change. Once setup is complete, they are again compared octet-for-octet.

To summarize non-normatively, the comparison steps including setup are:

1. If the domain contains A-labels, transform them to U-labels.

2. If the domain contains ASCII NR-LDH labels, lowercase them.

3. Compare strings octet-for-octet for equivalence.

This specification expressly does not define any wildcard characters and SmtpUTF8Mailbox comparison implementations MUST NOT interpret any character as wildcards. Instead, to specify multiple email addresses through SmtpUTF8Mailbox, the certificate MUST use multiple subjectAltNames or issuerAltNames to explicitly carry any additional email addresses.

6. Name constraints in path validation

This section updates Section 4.2.1.10 of [RFC5280] to extend rfc822Name name constraints to SmtpUTF8Mailbox subjectAltNames. A SmtpUTF8Mailbox aware path validators will apply name constraint comparison to the subject distinguished name and both forms of subject alternative name rfc822Name and SmtpUTF8Mailbox.

Both rfc822Name and SmtpUTF8Mailbox subject alternative names represent the same underlying email address namespace. Since legacy

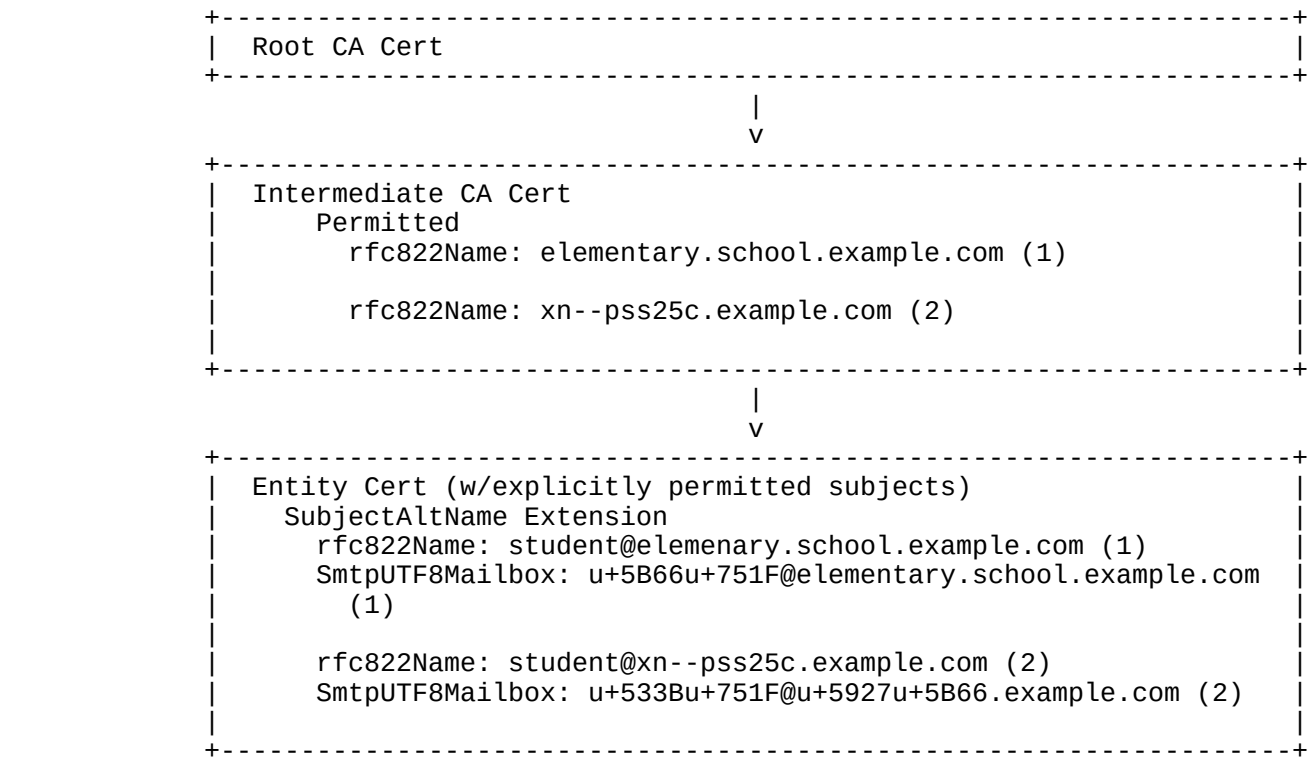
CAs constrained to issue certificates for a specific set of domains would lack corresponding UTF-8 constraints, [ID-lamps-rfc5280-i18n-update] updates modifies and extends rfc822Name name constraints defined in [RFC5280] to cover SmtpUTF8Mailbox subject alternative names. This ensures that the introduction of SmtpUTF8Mailbox does not violate existing name constraints. Since it is not valid to include non-ASCII UTF-8 characters in the local-part of rfc822Name name constraints, and since name constraints that include a local-part are rarely, if at all, used in practice, name constraints updated in [ID-lamps-rfc5280-i18n-update] admit the forms that represent all addresses at a host or all mailboxes in a domain, and deprecates rfc822Name name constraints that represent a particular mailbox. That is, rfc822Name constraints with a local-part SHOULD NOT be used.

Constraint comparison with SmtpUTF8Mailbox subjectAltName starts with the setup steps defined by Section 5. Setup converts the inputs of the comparison which is one of a subject distinguished name or a rfc822Name or SmtpUTF8Mailbox subjectAltName, and one of a rfc822Name name constraint, to constraint comparison form. For rfc822Name name constraint, this will convert any domain A-labels to U-labels. For both the name constraint and the subject, this will lower case any domain NR-LDH labels. Strip the local-part and "@" separator from each rfc822Name and SmtpUTF8Mailbox, leaving just the domain-part. After setup, this follows the comparison steps defined in 4.2.1.10 of [RFC5280] as follows. If the resulting name constraint domain starts with a "." character, then for the name constraint to match, a suffix of the resulting subject alternative name domain MUST match the name constraint (including the leading ".") octet for octet. If the resulting name constraint domain does not start with a "." character, then for the name constraint to match, the entire resulting subject alternative name domain MUST match the name constraint octet for octet.

Certificate Authorities that wish to issue CA certificates with email address name constraint MUST use rfc822Name subject alternative names only. These MUST be IDNA2008 conformant names with no mappings, and with non-ASCII domains encoded in A-labels only.

The name constraint requirement with SmtpUTF8Mailbox subject alternative name is illustrated in the non-normative diagram Figure 1. The first example (1) illustrates a permitted rfc822Name ASCII only hostname name constraint, and the corresponding valid rfc822Name subjectAltName and SmtpUTF8Mailbox subjectAltName email addresses. The second example (2) illustrates a permitted rfc822Name hostname name constraint with A-label, and the corresponding valid rfc822Name subjectAltName and SmtpUTF8Mailbox subjectAltName email addresses. Note that an email address with ASCII only local-part is

encoded as rfc822Name despite also having unicode present in the domain.



Name constraints with SmtpUTF8Name and rfc822Name

Figure 1

7. Security Considerations

Use of SmtpUTF8Mailbox for certificate subjectAltName (and issuerAltName) will incur many of the same security considerations as in Section 8 in [RFC5280], but introduces a new issue by permitting non-ASCII characters in the email address local-part. This issue, as mentioned in Section 4.4 of [RFC5890] and in Section 4 of [RFC6532], is that use of Unicode introduces the risk of visually similar and identical characters which can be exploited to deceive the recipient. The former document references some means to mitigate against these attacks. See [WEBER] for more background on security issues with Unicode.

8. IANA Considerations

In Section 3 and the ASN.1 module identifier defined in Appendix A. IANA is kindly requested to make the following assignments for:

The LAMPS-EaiAddresses-2016 ASN.1 module in the "SMI Security for PKIX Module Identifier" registry (1.3.6.1.5.5.7.0).

The SmtpUTF8Mailbox otherName in the "PKIX Other Name Forms" registry (1.3.6.1.5.5.7.8). {{ Note to IANA: id-on-smtputf8Name was assigned based on an earlier version of this document. Please change that entry to id-on-SmtpUTF8Mailbox. }}

9. References

9.1. Normative References

[ID-lamps-rfc5280-i18n-update]
Housley, R., "Internationalization Updates to RFC 5280", June 2017, <https://datatracker.ietf.org/doc/draft-housley-rfc5280-i18n-update/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[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, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <https://www.rfc-editor.org/info/rfc5321>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <https://www.rfc-editor.org/info/rfc5891>.

[RFC6530] Klensin, J. and Y. Ko, "Overview and Framework for Internationalized Email", RFC 6530, DOI 10.17487/RFC6530, February 2012, <https://www.rfc-editor.org/info/rfc6530>.

[RFC6531] Yao, J. and W. Mao, "SMTP Extension for Internationalized Email", RFC 6531, DOI 10.17487/RFC6531, February 2012, <https://www.rfc-editor.org/info/rfc6531>.

[RFC6532] Yang, A., Steele, S., and N. Freed, "Internationalized Email Headers", RFC 6532, DOI 10.17487/RFC6532, February 2012, <https://www.rfc-editor.org/info/rfc6532>.

9.2. Informative References

[RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <https://www.rfc-editor.org/info/rfc5912>.

[WEBER] Weber, C., "Attacking Software Globalization", March 2010, <https://www.lookout.net/files/Chris_Weber_Character%20Transformations%20v1.7_IUC33.pdf>.

Appendix A. ASN.1 Module

The following ASN.1 module normatively specifies the SmtpUTF8Mailbox structure. This specification uses the ASN.1 definitions from [RFC5912] with the 2002 ASN.1 notation used in that document. [RFC5912] updates normative documents using older ASN.1 notation.

```
  LAMPS-EaiAddresses-2016
    { iso(1) identified-organization(3) dod(6)
      internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-lamps-eai-addresses-2016(TBD) }

  DEFINITIONS IMPLICIT TAGS ::=
  BEGIN

  IMPORTS
    OTHER-NAME
    FROM PKIX1Implicit-2009
      { iso(1) identified-organization(3) dod(6) internet(1) security(5)
      mechanisms(5) pkix(7) id-mod(0) id-mod-pkix1-implicit-02(59) }

    id-pkix
    FROM PKIX1Explicit-2009
      { iso(1) identified-organization(3) dod(6) internet(1) security(5)
      mechanisms(5) pkix(7) id-mod(0) id-mod-pkix1-explicit-02(51) } ;

  --
  -- otherName carries additional name types for subjectAltName,
  -- issuerAltName, and other uses of GeneralNames.
  --

    id-on OBJECT IDENTIFIER ::= { id-pkix 8 }

    SmtpUtf8OtherNames OTHER-NAME ::= { on-SmtpUTF8Mailbox, ... }

    on-SmtpUTF8Mailbox OTHER-NAME ::= {
        SmtpUTF8Mailbox IDENTIFIED BY id-on-SmtpUTF8Mailbox
    }

    id-on-SmtpUTF8Mailbox OBJECT IDENTIFIER ::= { id-on 9 }

    SmtpUTF8Mailbox ::= UTF8String (SIZE (1..MAX))
     -- SmtpUTF8Mailbox conforms to Mailbox as specified
     -- in Section 3.3 of RFC 6531.

  END
```

Appendix B. Example of SmtpUTF8Mailbox

This non-normative example demonstrates using SmtpUTF8Mailbox as an otherName in GeneralName to encode the email address "u+8001u+5E2B@example.com".

```
The hexadecimal DER encoding of the email address is:
A022060A 2B060105 05070012 0809A014 0C12E880 81E5B8AB 40657861
6D706C65 2E636F6D

The text decoding is:
  0  34: [0] {
  2  10:   OBJECT IDENTIFIER '1 3 6 1 5 5 7 0 18 8 9'
 14  20:   [0] {
 16  18:     UTF8String '..@example.com'
       :     }
       :   }
```

Figure 2

The example was encoded on the OSS Nokalva ASN.1 Playground and the above text decoding is an output of Peter Gutmann's "dumpasn1" program.

Appendix C. Acknowledgements

Thank you to Magnus Nystrom for motivating this document. Thanks to Russ Housley, Nicolas Lidzborski, Laetitia Baudoin, Ryan Sleevi, Sean Leonard, Sean Turner, John Levine, and Patrik Falstrom for their feedback. Also special thanks to John Klensin for his valuable input on internationalization, Unicode and ABNF formatting, to Jim Schaad for his help with the ASN.1 example and his helpful feedback, and especially to Viktor Dukhovni for helping us with name constraints and his many detailed document reviews.

Authors' Addresses

Alexey Melnikov (editor)
Isode Ltd
14 Castle Mews
Hampton, Middlesex TW12 2NP
UK

Email: Alexey.Melnikov@isode.com

Weihaw Chuang (editor)
Google, Inc.
1600 Amphitheater Parkway
Mountain View, CA 94043
US

Email: weihaw@google.com

LAMPS
Internet-Draft
Obsoletes: 5750 (if approved)
Intended status: Standards Track
Expires: March 8, 2019

J. Schaad
August Cellars
B. Ramsdell
Brute Squad Labs, Inc.
S. Turner
sn3rd

September 4, 2018

Secure/Multipurpose Internet Mail Extensions (S/ MIME) Version 4.0 Certificate Handling
draft-ietf-lamps-rfc5750-bis-08

Abstract

This document specifies conventions for X.509 certificate usage by Secure/Multipurpose Internet Mail Extensions (S/MIME) v4.0 agents. S/MIME provides a method to send and receive secure MIME messages, and certificates are an integral part of S/MIME agent processing. S/MIME agents validate certificates as described in RFC 5280, the Internet X.509 Public Key Infrastructure Certificate and CRL Profile. S/MIME agents must meet the certificate processing requirements in this document as well as those in RFC 5280. This document obsoletes RFC 5750.

Contributing to this document

The source for this draft is being maintained in GitHub. Suggested changes should be submitted as pull requests at <https://github.com/lamps-wg/smime>. Instructions are on that page as well. Editorial changes can be managed in GitHub, but any substantial issues need to be discussed on the LAMPS mailing list.

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 https://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 March 8, 2019.

Copyright Notice

Copyright (c) 2018 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 (https://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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction

S/MIME (Secure/Multipurpose Internet Mail Extensions) v4.0, described in [I-D.ietf-lamps-rfc5751-bis], provides a method to send and receive secure MIME messages. Before using a public key to provide security services, the S/MIME agent MUST verify that the public key is valid. S/MIME agents MUST use PKIX certificates to validate public keys as described in the Internet X.509 Public Key Infrastructure (PKIX) Certificate and CRL Profile [RFC5280]. S/MIME agents MUST meet the certificate processing requirements documented in this document in addition to those stated in [RFC5280].

This specification is compatible with the Cryptographic Message Syntax (CMS) RFC 5652 [RFC5652] in that it uses the data types defined by CMS. It also inherits all the varieties of architectures for certificate-based key management supported by CMS.

This document obsoletes [RFC5750]. The most significant changes revolve around changes in recommendations around the cryptographic algorithms used by the specification. More details can be found in Section 1.6.

1.1. Definitions

For the purposes of this document, the following definitions apply.

ASN.1: Abstract Syntax Notation One, as defined in ITU-T X.680 [X.680].

Attribute certificate (AC): An X.509 AC is a separate structure from a subject's public key X.509 certificate. A subject may have

multiple X.509 ACs associated with each of its public key X.509 certificates. Each X.509 AC binds one or more attributes with one of the subject's public key X.509 certificates. The X.509 AC syntax is defined in [RFC5755].

Certificate: A type that binds an entity's name to a public key with a digital signature. This type is defined in the Internet X.509 Public Key Infrastructure (PKIX) Certificate and CRL Profile [RFC5280]. This type also contains the distinguished name of the certificate issuer (the signer), an issuer-specific serial number, the issuer's signature algorithm identifier, a validity period, and extensions also defined in that document.

Certificate Revocation List (CRL): A type that contains information about certificates whose validity an issuer has revoked. The information consists of an issuer name, the time of issue, the next scheduled time of issue, a list of certificate serial numbers and their associated revocation times, and extensions as defined in [RFC5280]. The CRL is signed by the issuer. The type intended by this specification is the one defined in [RFC5280].

Receiving agent: Software that interprets and processes S/MIME CMS objects, MIME body parts that contain CMS objects, or both.

Sending agent: Software that creates S/MIME CMS objects, MIME body parts that contain CMS objects, or both.

S/MIME agent: User software that is a receiving agent, a sending agent, or both.

1.2. Conventions Used 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

We define the additional requirement levels:

SHOULD+ This term means the same as SHOULD. However, the authors expect that a requirement marked as SHOULD+ will be promoted at some future time to be a MUST.

SHOULD- This term means the same as SHOULD. However, the authors expect that a requirement marked as SHOULD- will be demoted to a MAY in a future version of this document.

MUST- This term means the same as MUST. However, the authors expect that this requirement will no longer be a MUST in a future document. Although its status will be determined at a later time, it is reasonable to expect that if a future revision of a document alters the status of a MUST-requirement, it will remain at least a SHOULD or a SHOULD-.

The term RSA in this document almost always refers to the PKCS#1 v1.5 RSA signature algorithm even when not qualified as such. There are a couple of places where it refers to the general RSA cryptographic operation; these can be determined from the context where it is used.

1.3. Compatibility with Prior Practice S/MIME

S/MIME version 4.0 agents ought to attempt to have the greatest interoperability possible with agents for prior versions of S/MIME.

S/MIME version 2 is described in RFC 2311 through RFC 2315 inclusive [SMIMEv2], S/MIME version 3 is described in RFC 2630 through RFC 2634 inclusive and RFC 5035 [SMIMEv3], and S/MIME version 3.1 is described in RFC 3850, RFC 3851, RFC 3852, RFC 2634, and RFC 5035 [SMIMEv3.1]. RFC 2311 also has historical information about the development of S/MIME.

Appendix A contains information about algorithms that were used for prior versions of S/MIME but are no longer considered to meet modern security standards. Support of these algorithms may be needed to support historic S/MIME artifacts such as messages or files, but SHOULD NOT be used for new artifacts.

1.4. Changes from S/MIME v3 to S/MIME v3.1

This section reflects the changes that were made when S/MIME v3.1 was released. The RFC2119 langauage may have superceeded in later versions.

Version 1 and version 2 CRLs MUST be supported.

Multiple certification authority (CA) certificates with the same subject and public key, but with overlapping validity periods, MUST be supported.

Version 2 attribute certificates SHOULD be supported, and version 1 attributes certificates MUST NOT be used.

The use of the MD2 digest algorithm for certificate signatures is discouraged, and security language was added.

Clarified use of email address use in certificates. Certificates that do not contain an email address have no requirements for verifying the email address associated with the certificate.

Receiving agents SHOULD display certificate information when displaying the results of signature verification.

Receiving agents MUST NOT accept a signature made with a certificate that does not have at least one of the the digitalSignature or nonRepudiation bits set.

Clarifications for the interpretation of the key usage and extended key usage extensions.

1.5. Changes from S/MIME v3.1 to S/MIME v3.2

This section reflects the changes that were made when S/MIME v3.2 was released. The RFC2119 langauage may have superceeded in later versions.

Conventions Used in This Document: Moved to Section 1.2. Added definitions for SHOULD+, SHOULD-, and MUST-.

Section 1.1: Updated ASN.1 definition and reference.

Section 1.3: Added text about v3.1 RFCs.

Section 3: Aligned email address text with RFC 5280. Updated note to indicate emailAddress IA5String upper bound is 255 characters. Added text about matching email addresses.

Section 4.2: Added text to indicate how S/MIME agents locate the correct user certificate.

Section 4.3: RSA with SHA-256 (PKCS #1 v1.5) added as MUST; DSA with SHA-256 added as SHOULD+; RSA with SHA-1, DSA with SHA-1, and RSA with MD5 changed to SHOULD-; and RSASSA-PSS with SHA-256 added as SHOULD+. Updated key sizes and changed pointer to PKIX RFCs.

Section 4.4.1: Aligned with PKIX on use of basic constraints extension in CA certificates. Clarified which extension is used to constrain end entities from using their keys to perform issuing authority operations.

Section 5: Updated security considerations.

Section 7: Moved references from Appendix B to Section 6. Updated the references.

Appendix A: Moved Appendix A to Appendix B. Added Appendix A to move S/MIME v2 Certificate Handling to Historic Status.

1.6. Changes since S/MIME 3.2

This section reflects the changes that were made when S/MIME v4.0 was released. The RFC2119 langauage may have superceeded in later versions.

Section 3: Require support for internationalized email addresses.

Section 4.3: Mandated support for ECDSA with P-256 and Ed25519. Moved algorithms with SHA-1 and MD5 to historical status. Moved DSA support to historical status. Increased lower bounds on RSA key sizes.

Appendix A: Add a new appendix for algorithms that are now considered to be historical.

2. CMS Options

The CMS message format allows for a wide variety of options in content and algorithm support. This section puts forth a number of support requirements and recommendations in order to achieve a base level of interoperability among all S/MIME implementations. Most of the CMS format for S/MIME messages is defined in [I-D.ietf-lamps-rfc5751-bis].

2.1. Certificate Revocation Lists

Receiving agents MUST support the Certificate Revocation List (CRL) format defined in [RFC5280]. If sending agents include CRLs in outgoing messages, the CRL format defined in [RFC5280] MUST be used. Receiving agents MUST support both v1 and v2 CRLs.

All agents MUST be capable of performing revocation checks using CRLs as specified in [RFC5280]. All agents MUST perform revocation status checking in accordance with [RFC5280]. Receiving agents MUST recognize CRLs in received S/MIME messages.

Agents SHOULD store CRLs received in messages for use in processing later messages.

2.2. Certificate Choices

Receiving agents MUST support v1 X.509 and v3 X.509 certificates as profiled in [RFC5280]. End-entity certificates MAY include an Internet mail address, as described in Section 3.

Receiving agents SHOULD support X.509 version 2 attribute certificates. See [RFC5755] for details about the profile for attribute certificates.

2.2.1. Historical Note about CMS Certificates

The CMS message format supports a choice of certificate formats for public key content types: PKIX, PKCS #6 extended certificates [PKCS6], and PKIX attribute certificates.

The PKCS #6 format is not in widespread use. In addition, PKIX certificate extensions address much of the same functionality and flexibility as was intended in the PKCS #6. Thus, sending and receiving agents MUST NOT use PKCS #6 extended certificates. Receiving agents MUST be able to parse and process a message containing PKCS #6 extended certificates although ignoring those certificates is expected behavior.

X.509 version 1 attribute certificates are also not widely implemented, and have been superseded with version 2 attribute certificates. Sending agents MUST NOT send version 1 attribute certificates.

2.3. CertificateSet

Receiving agents MUST be able to handle an arbitrary number of certificates of arbitrary relationship to the message sender and to each other in arbitrary order. In many cases, the certificates included in a signed message may represent a chain of certification from the sender to a particular root. There may be, however, situations where the certificates in a signed message may be unrelated and included for convenience.

Sending agents SHOULD include any certificates for the user’s public key(s) and associated issuer certificates. This increases the likelihood that the intended recipient can establish trust in the originator’s public key(s). This is especially important when sending a message to recipients that may not have access to the sender’s public key through any other means or when sending a signed message to a new recipient. The inclusion of certificates in outgoing messages can be omitted if S/MIME objects are sent within a group of correspondents that has established access to each other’s

certificates by some other means such as a shared directory or manual certificate distribution. Receiving S/MIME agents SHOULD be able to handle messages without certificates by using a database or directory lookup scheme to find them.

A sending agent SHOULD include at least one chain of certificates up to, but not including, a certification authority (CA) that it believes that the recipient may trust as authoritative. A receiving agent MUST be able to handle an arbitrarily large number of certificates and chains.

Agents MAY send CA certificates, that is, cross-certificates, self-issued certificates, and self-signed certificates. Note that receiving agents SHOULD NOT simply trust any self-signed certificates as valid CAs, but SHOULD use some other mechanism to determine if this is a CA that should be trusted. Also note that when certificates contain Digital Signature Algorithm (DSA) public keys the parameters may be located in the root certificate. This would require that the recipient possess both the end-entity certificate and the root certificate to perform a signature verification, and is a valid example of a case where transmitting the root certificate may be required.

Receiving agents MUST support chaining based on the distinguished name fields. Other methods of building certificate chains MAY be supported.

Receiving agents SHOULD support the decoding of X.509 attribute certificates included in CMS objects. All other issues regarding the generation and use of X.509 attribute certificates are outside of the scope of this specification. One specification that addresses attribute certificate use is defined in [RFC3114].

3. Using Distinguished Names for Internet Mail

End-entity certificates MAY contain an Internet mail address. Email addresses restricted to 7-bit ASCII characters use the pkcs-9-at-emailAddress OID (see below) and are encoded as described in Section 4.2.1.6 of [RFC5280]. Internationalized Email address names use the OID defined in [I-D.ietf-lamps-eai-addresses] and are encoded as described there. The email address SHOULD be in the subjectAltName extension, and SHOULD NOT be in the subject distinguished name.

Receiving agents MUST recognize and accept certificates that contain no email address. Agents are allowed to provide an alternative mechanism for associating an email address with a certificate that does not contain an email address, such as through the use of the

agent's address book, if available. Receiving agents MUST recognize both ASCII and internationalized email addresses in the subjectAltName field. Receiving agents MUST recognize email addresses in the Distinguished Name field in the PKCS #9 [RFC2985] emailAddress attribute:

```
pkcs-9-at-emailAddress OBJECT IDENTIFIER ::=
 { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) 1 }
```

Note that this attribute MUST be encoded as IA5String and has an upper bound of 255 characters. Comparing of email addresses is fraught with peril. [I-D.ietf-lamps-eai-addresses] defines the procedure for doing comparison of Internationalized email addresses. For ASCII email addresses the domain component (right-hand side of the '@') MUST be compared using a case-insensitive function. The local name component (left-hand side of the '@') SHOULD be compared using a case-insensitive function. Some localities may perform other transformations on the local name component before doing the comparison, however an S/MIME client cannot know what specific localities do.

Sending agents SHOULD make the address in the From or Sender header in a mail message match an Internet mail address in the signer's certificate. Receiving agents MUST check that the address in the From or Sender header of a mail message matches an Internet mail address in the signer's certificate, if mail addresses are present in the certificate. A receiving agent SHOULD provide some explicit alternate processing of the message if this comparison fails; this might be done by displaying or logging a message that shows the recipient the mail addresses in the certificate or other certificate details.

A receiving agent SHOULD display a subject name or other certificate details when displaying an indication of successful or unsuccessful signature verification.

All subject and issuer names MUST be populated (i.e., not an empty SEQUENCE) in S/MIME-compliant X.509 certificates, except that the subject distinguished name (DN) in a user's (i.e., end-entity) certificate MAY be an empty SEQUENCE in which case the subjectAltName extension will include the subject's identifier and MUST be marked as critical.

4. Certificate Processing

S/MIME agents need to provide some certificate retrieval mechanism in order to gain access to certificates for recipients of digital envelopes. There are many ways to implement certificate retrieval

mechanisms. [X.500] directory service is an excellent example of a certificate retrieval-only mechanism that is compatible with classic X.500 Distinguished Names. The IETF has published [RFC8162] which describes an experimental protocol to retrieve certificates from the Domain Name System (DNS). Until such mechanisms are widely used, their utility may be limited by the small number of the correspondent's certificates that can be retrieved. At a minimum, for initial S/MIME deployment, a user agent could automatically generate a message to an intended recipient requesting the recipient's certificate in a signed return message.

Receiving and sending agents SHOULD also provide a mechanism to allow a user to "store and protect" certificates for correspondents in such a way so as to guarantee their later retrieval. In many environments, it may be desirable to link the certificate retrieval/ storage mechanisms together in some sort of certificate database. In its simplest form, a certificate database would be local to a particular user and would function in a similar way as an "address book" that stores a user's frequent correspondents. In this way, the certificate retrieval mechanism would be limited to the certificates that a user has stored (presumably from incoming messages). A comprehensive certificate retrieval/storage solution might combine two or more mechanisms to allow the greatest flexibility and utility to the user. For instance, a secure Internet mail agent might resort to checking a centralized certificate retrieval mechanism for a certificate if it cannot be found in a user's local certificate storage/retrieval database.

Receiving and sending agents SHOULD provide a mechanism for the import and export of certificates, using a CMS certs-only message. This allows for import and export of full certificate chains as opposed to just a single certificate. This is described in [RFC5751].

Agents MUST handle multiple valid certification authority (CA) certificates containing the same subject name and the same public keys but with overlapping validity intervals.

4.1. Certificate Revocation Lists

In general, it is always better to get the latest CRL information from a CA than to get information stored in an incoming messages. A receiving agent SHOULD have access to some CRL retrieval mechanism in order to gain access to certificate revocation information when validating certification paths. A receiving or sending agent SHOULD also provide a mechanism to allow a user to store incoming certificate revocation information for correspondents in such a way so as to guarantee its later retrieval.

Receiving and sending agents SHOULD retrieve and utilize CRL information every time a certificate is verified as part of a certification path validation even if the certificate was already verified in the past. However, in many instances (such as off-line verification) access to the latest CRL information may be difficult or impossible. The use of CRL information, therefore, may be dictated by the value of the information that is protected. The value of the CRL information in a particular context is beyond the scope of this specification but may be governed by the policies associated with particular certification paths.

All agents MUST be capable of performing revocation checks using CRLs as specified in [RFC5280]. All agents MUST perform revocation status checking in accordance with [RFC5280]. Receiving agents MUST recognize CRLs in received S/MIME messages.

4.2. Certificate Path Validation

In creating a user agent for secure messaging, certificate, CRL, and certification path validation should be highly automated while still acting in the best interests of the user. Certificate, CRL, and path validation MUST be performed as per [RFC5280] when validating a correspondent's public key. This is necessary before using a public key to provide security services such as verifying a signature, encrypting a content-encryption key (e.g., RSA), or forming a pairwise symmetric key (e.g., Diffie-Hellman) to be used to encrypt or decrypt a content-encryption key.

Certificates and CRLs are made available to the path validation procedure in two ways: a) incoming messages, and b) certificate and CRL retrieval mechanisms. Certificates and CRLs in incoming messages are not required to be in any particular order nor are they required to be in any way related to the sender or recipient of the message (although in most cases they will be related to the sender). Incoming certificates and CRLs SHOULD be cached for use in path validation and optionally stored for later use. This temporary certificate and CRL cache SHOULD be used to augment any other certificate and CRL retrieval mechanisms for path validation on incoming signed messages.

When verifying a signature and the certificates that are included in the message, if a signingCertificate attribute from RFC 2634 [ESS] or a signingCertificateV2 attribute from RFC 5035 [ESS] is found in an S/MIME message, it SHALL be used to identify the signer's certificate. Otherwise, the certificate is identified in an S/MIME message, either using the issuerAndSerialNumber, which identifies the signer's certificate by the issuer's distinguished name and the

certificate serial number, or the subjectKeyIdentifier, which identifies the signer's certificate by a key identifier.

When decrypting an encrypted message, if a SMIMEEncryptionKeyPreference attribute is found in an encapsulating SignedData, it SHALL be used to identify the originator's certificate found in OriginatorInfo. See [RFC5652] for the CMS fields that reference the originator's and recipient's certificates.

4.3. Certificate and CRL Signing Algorithms and Key Sizes

Certificates and Certificate Revocation Lists (CRLs) are signed by the certificate issuer. Receiving agents:

- MUST support ECDSA with curve P-256 with SHA-256.

- MUST support EdDSA with curve 25519 using PureEdDSA mode.

- MUST- support RSA PKCS#1 v1.5 with SHA-256.

- SHOULD support RSASSA-PSS with SHA-256.

Implementations SHOULD use deterministic generation for the parameter 'k' for ECDSA as outlined in [RFC6979]. EdDSA is defined to generate this parameter deterministically.

The following are the RSA and RSASSA-PSS key size requirements for S/MIME receiving agents during certificate and CRL signature verification:

```
        key size <= 2047 : SHOULD NOT (see Historic Considerations)
2048 <= key size <= 4096 : MUST (see Security Considerations)
4096 <  key size         : MAY  (see Security Considerations)
```

The signature algorithm object identifiers for RSA PKCS#1 v1.5 and RSASSA-PSS with SHA-256 using 1024-bit through 3072-bit public keys are specified in [RFC4055] and the signature algorithm definition is found in [FIPS186-2] with Change Notice 1.

The signature algorithm object identifiers for RSA PKCS#1 v1.5 and RSASSA-PSS with SHA-256 using 4096-bit public keys are specified in [RFC4055] and the signature algorithm definition is found in [RFC3447].

For RSASSA-PSS with SHA-256 see [RFC4056].

For ECDSA see [RFC5758] and [RFC6090]. The first reference provides the signature algorithm's object identifier and the second provides

the signature algorithm's definition. Curves other than curve P-256 MAY be used as well.

For EdDSA see [I-D.ietf-curdle-pkix] and [RFC8032]. The first reference provides the signature algorithm's object identifier and the second provides the signature algorithm's definition. Other curves than curve 25519 MAY be used as well.

4.4. PKIX Certificate Extensions

PKIX describes an extensible framework in which the basic certificate information can be extended and describes how such extensions can be used to control the process of issuing and validating certificates. The LAMPS Working Group has ongoing efforts to identify and create extensions that have value in particular certification environments. Further, there are active efforts underway to issue PKIX certificates for business purposes. This document identifies the minimum required set of certificate extensions that have the greatest value in the S/MIME environment. The syntax and semantics of all the identified extensions are defined in [RFC5280].

Sending and receiving agents MUST correctly handle the basic constraints, key usage, authority key identifier, subject key identifier, and subject alternative names certificate extensions when they appear in end-entity and CA certificates. Some mechanism SHOULD exist to gracefully handle other certificate extensions when they appear in end-entity or CA certificates.

Certificates issued for the S/MIME environment SHOULD NOT contain any critical extensions (extensions that have the critical field set to TRUE) other than those listed here. These extensions SHOULD be marked as non-critical unless the proper handling of the extension is deemed critical to the correct interpretation of the associated certificate. Other extensions may be included, but those extensions SHOULD NOT be marked as critical.

Interpretation and syntax for all extensions MUST follow [RFC5280], unless otherwise specified here.

4.4.1. Basic Constraints

The basic constraints extension serves to delimit the role and position that an issuing authority or end-entity certificate plays in a certification path.

For example, certificates issued to CAs and subordinate CAs contain a basic constraints extension that identifies them as issuing authority certificates. End-entity certificates contain the key usage

extension that restrains end-entities from using the key when performing issuing authority operations (see Section 4.4.2).

As per [RFC5280], certificates MUST contain a basicConstraints extension in CA certificates, and SHOULD NOT contain that extension in end-entity certificates.

4.4.2. Key Usage Certificate Extension

The key usage extension serves to limit the technical purposes for which a public key listed in a valid certificate may be used. Issuing authority certificates may contain a key usage extension that restricts the key to signing certificates, certificate revocation lists, and other data.

For example, a certification authority may create subordinate issuer certificates that contain a key usage extension that specifies that the corresponding public key can be used to sign end user certificates and sign CRLs.

If a key usage extension is included in a PKIX certificate, then it MUST be marked as critical.

S/MIME receiving agents MUST NOT accept the signature of a message if it was verified using a certificate that contains the key usage extension without at least one of the digitalSignature or nonRepudiation bits set. Sometimes S/MIME is used as a secure message transport for applications beyond interpersonal messaging; in such cases, the S/MIME-enabled application can specify additional requirements concerning the digitalSignature or nonRepudiation bits within this extension.

If the key usage extension is not specified, receiving clients MUST presume that both the digitalSignature and nonRepudiation bits are set.

4.4.3. Subject Alternative Name

The subject alternative name extension is used in S/MIME as the preferred means to convey the email address(es) that correspond(s) to the entity for this certificate. If the local portion of the email address is ASCII, it MUST be encoded using the rfc822Name CHOICE of the GeneralName type as described in [RFC5280], Section 4.2.1.6. If the local portion of the email address is not ASCII, it MUST be encoded using the otherName CHOICE of the GeneralName type as described in [I-D.ietf-lamps-eai-addresses], Section 3. Since the SubjectAltName type is a SEQUENCE OF GeneralName, multiple email addresses MAY be present.

4.4.4. Extended Key Usage Extension

The extended key usage extension also serves to limit the technical purposes for which a public key listed in a valid certificate may be used. The set of technical purposes for the certificate therefore are the intersection of the uses indicated in the key usage and extended key usage extensions.

For example, if the certificate contains a key usage extension indicating digital signature and an extended key usage extension that includes the email protection OID, then the certificate may be used for signing but not encrypting S/MIME messages. If the certificate contains a key usage extension indicating digital signature but no extended key usage extension, then the certificate may also be used to sign but not encrypt S/MIME messages.

If the extended key usage extension is present in the certificate, then interpersonal message S/MIME receiving agents MUST check that it contains either the emailProtection or the anyExtendedKeyUsage OID as defined in [RFC5280]. S/MIME uses other than interpersonal messaging MAY require the explicit presence of the extended key usage extension or other OIDs to be present in the extension or both.

5. IANA Considertions

This document has no new IANA considerations.

6. Security Considerations

All of the security issues faced by any cryptographic application must be faced by a S/MIME agent. Among these issues are protecting the user’s private key, preventing various attacks, and helping the user avoid mistakes such as inadvertently encrypting a message for the wrong recipient. The entire list of security considerations is beyond the scope of this document, but some significant concerns are listed here.

When processing certificates, there are many situations where the processing might fail. Because the processing may be done by a user agent, a security gateway, or other program, there is no single way to handle such failures. Just because the methods to handle the failures have not been listed, however, the reader should not assume that they are not important. The opposite is true: if a certificate is not provably valid and associated with the message, the processing software should take immediate and noticeable steps to inform the end user about it.

Some of the many places where signature and certificate checking might fail include:

- no Internet mail addresses in a certificate match the sender of a message, if the certificate contains at least one mail address
- no certificate chain leads to a trusted CA
- no ability to check the CRL for a certificate
- an invalid CRL was received
- the CRL being checked is expired
- the certificate is expired
- the certificate has been revoked

There are certainly other instances where a certificate may be invalid, and it is the responsibility of the processing software to check them all thoroughly, and to decide what to do if the check fails.

It is possible for there to be multiple unexpired CRLs for a CA. If an agent is consulting CRLs for certificate validation, it SHOULD make sure that the most recently issued CRL for that CA is consulted, since an S/MIME message sender could deliberately include an older unexpired CRL in an S/MIME message. This older CRL might not include recently revoked certificates, which might lead an agent to accept a certificate that has been revoked in a subsequent CRL.

When determining the time for a certificate validity check, agents have to be careful to use a reliable time. In most cases the time used SHOULD be the current time, some exceptions to this would be:

- The time the message was received is stored in a secure manner and is used at a later time to validate the message.
- The time in a SigningTime attribute found in a counter signature attribute which has been successfully validated.

The SigningTime attribute could be deliberately set to direct the receiving agent to check a CRL that could have out-of-date revocation status for a certificate, or cause an improper result when checking the Validity field of a certificate. This could be done either by the sender of the message, or an attacker which has compromised the key of the sender.

In addition to the Security Considerations identified in [RFC5280], caution should be taken when processing certificates that have not first been validated to a trust anchor. Certificates could be manufactured by untrusted sources for the purpose of mounting denial of service or other attacks. For example, keys selected to require excessive cryptographic processing, or extensive lists of CRL Distribution Point (CDP) and/or Authority Information Access (AIA) addresses in the certificate, could be used to mount denial-of-service attacks. Similarly, attacker-specified CDP and/or AIA addresses could be included in fake certificates to allow the originator to detect receipt of the message even if signature verification fails.

RSA keys of less than 2048 bits are now considered by many experts to be cryptographically insecure (due to advances in computing power), and SHOULD no longer be used to sign certificates or CRLs. Such keys were previously considered secure, so processing previously received signed and encrypted mail may require processing certificates or CRLs signed with weak keys. Implementations that wish to support previous versions of S/MIME or process old messages need to consider the security risks that result from accepting certificates and CRLs with smaller key sizes (e.g., spoofed certificates) versus the costs of denial of service. If an implementation supports verification of certificates or CRLs generated with RSA and DSA keys of less than 2048 bits, it MUST warn the user. Implementers should consider providing a stronger warning for weak signatures on certificates and CRLs associated with newly received messages than the one provided for certificates and CRLs associated with previously stored messages. Server implementations (e.g., secure mail list servers) where user warnings are not appropriate SHOULD reject messages with weak cryptography.

If an implementation is concerned about compliance with National Institute of Standards and Technology (NIST) key size recommendations, then see [SP800-57].

7. References

7.1. Normative References

[FIPS186-2]
National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS) [With Change Notice 1]", Federal Information Processing Standards Publication 186-2, January 2000.

[FIPS186-3]
National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", Federal Information Processing Standards Publication 186-3, June 2009.

[I-D.ietf-lamps-eai-addresses]
Melnikov, A. and W. Chuang, "Internationalized Email Addresses in X.509 certificates", draft-ietf-lamps-eai-addresses-18 (work in progress), March 2018.

[I-D.ietf-lamps-rfc5751-bis]
Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", draft-ietf-lamps-rfc5751-bis-11 (work in progress), July 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2634] Hoffman, P., Ed., "Enhanced Security Services for S/MIME", RFC 2634, DOI 10.17487/RFC2634, June 1999, <https://www.rfc-editor.org/info/rfc2634>.

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, DOI 10.17487/RFC2985, November 2000, <https://www.rfc-editor.org/info/rfc2985>.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, DOI 10.17487/RFC3279, April 2002, <https://www.rfc-editor.org/info/rfc3279>.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, DOI 10.17487/RFC3447, February 2003, <https://www.rfc-editor.org/info/rfc3447>.

[RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, DOI 10.17487/RFC4055, June 2005, <https://www.rfc-editor.org/info/rfc4055>.

[RFC4056] Schaad, J., "Use of the RSASSA-PSS Signature Algorithm in Cryptographic Message Syntax (CMS)", RFC 4056, DOI 10.17487/RFC4056, June 2005, <https://www.rfc-editor.org/info/rfc4056>.

[RFC5035] Schaad, J., "Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility", RFC 5035, DOI 10.17487/RFC5035, August 2007, <https://www.rfc-editor.org/info/rfc5035>.

[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, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5750] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling", RFC 5750, DOI 10.17487/RFC5750, January 2010, <https://www.rfc-editor.org/info/rfc5750>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <https://www.rfc-editor.org/info/rfc5751>.

[RFC5755] Farrell, S., Housley, R., and S. Turner, "An Internet Attribute Certificate Profile for Authorization", RFC 5755, DOI 10.17487/RFC5755, January 2010, <https://www.rfc-editor.org/info/rfc5755>.

[RFC5758] Dang, Q., Santesson, S., Moriarty, K., Brown, D., and T. Polk, "Internet X.509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA", RFC 5758, DOI 10.17487/RFC5758, January 2010, <https://www.rfc-editor.org/info/rfc5758>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[SMIMEv3.2] "S/MIME version 3.2".

This group of documents represents S/MIME version 3.2. This set of documents are [RFC2634], [RFC5750], [[This Document]], [RFC5652], and [RFC5035].

[SMIMEv4.0] "S/MIME version 4.0".

This group of documents represents S/MIME version 4.0. This set of documents are [RFC2634], [I-D.ietf-lamps-rfc5751-bis], [[This Document]], [RFC5652], and [RFC5035].

[X.680] "Information Technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation. ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002.".

7.2. Informational References

[ESS] "Enhanced Security Services for S/ MIME".

This is the set of documents dealing with enhanced security services and refers to [RFC2634] and [RFC5035].

[I-D.ietf-curdle-pkix] Josefsson, S. and J. Schaad, "Algorithm Identifiers for Ed25519, Ed448, X25519 and X448 for use in the Internet X.509 Public Key Infrastructure", draft-ietf-curdle-pkix-10 (work in progress), May 2018.

[PKCS6] RSA Laboratories, "PKCS #6: Extended-Certificate Syntax Standard", November 1993.

[RFC2311] Dusse, S., Hoffman, P., Ramsdell, B., Lundblade, L., and L. Repka, "S/MIME Version 2 Message Specification", RFC 2311, DOI 10.17487/RFC2311, March 1998, <https://www.rfc-editor.org/info/rfc2311>.

[RFC2312] Dusse, S., Hoffman, P., Ramsdell, B., and J. Weinstein, "S/MIME Version 2 Certificate Handling", RFC 2312, DOI 10.17487/RFC2312, March 1998, <https://www.rfc-editor.org/info/rfc2312>.

[RFC2313] Kaliski, B., "PKCS #1: RSA Encryption Version 1.5", RFC 2313, DOI 10.17487/RFC2313, March 1998, <https://www.rfc-editor.org/info/rfc2313>.

[RFC2314] Kaliski, B., "PKCS #10: Certification Request Syntax Version 1.5", RFC 2314, DOI 10.17487/RFC2314, March 1998, <https://www.rfc-editor.org/info/rfc2314>.

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <https://www.rfc-editor.org/info/rfc2315>.

[RFC2630] Housley, R., "Cryptographic Message Syntax", RFC 2630, DOI 10.17487/RFC2630, June 1999, <https://www.rfc-editor.org/info/rfc2630>.

[RFC2631] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, DOI 10.17487/RFC2631, June 1999, <https://www.rfc-editor.org/info/rfc2631>.

[RFC2632] Ramsdell, B., Ed., "S/MIME Version 3 Certificate Handling", RFC 2632, DOI 10.17487/RFC2632, June 1999, <https://www.rfc-editor.org/info/rfc2632>.

[RFC2633] Ramsdell, B., Ed., "S/MIME Version 3 Message Specification", RFC 2633, DOI 10.17487/RFC2633, June 1999, <https://www.rfc-editor.org/info/rfc2633>.

[RFC3114] Nicolls, W., "Implementing Company Classification Policy with the S/MIME Security Label", RFC 3114, DOI 10.17487/RFC3114, May 2002, <https://www.rfc-editor.org/info/rfc3114>.

[RFC3850] Ramsdell, B., Ed., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling", RFC 3850, DOI 10.17487/RFC3850, July 2004, <https://www.rfc-editor.org/info/rfc3850>.

[RFC3851] Ramsdell, B., Ed., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, DOI 10.17487/RFC3851, July 2004, <https://www.rfc-editor.org/info/rfc3851>.

[RFC3852] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, DOI 10.17487/RFC3852, July 2004, <https://www.rfc-editor.org/info/rfc3852>.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <https://www.rfc-editor.org/info/rfc6090>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <https://www.rfc-editor.org/info/rfc6151>.

[RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <https://www.rfc-editor.org/info/rfc6194>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8162] Hoffman, P. and J. Schlyter, "Using Secure DNS to Associate Certificates with Domain Names for S/MIME", RFC 8162, DOI 10.17487/RFC8162, May 2017, <https://www.rfc-editor.org/info/rfc8162>.

[SMIMEv2] "S/MIME version v2".

This group of documents represents S/MIME version 2. This set of documents are [RFC2311], [RFC2312], [RFC2313], [RFC2314], and [RFC2315].

[SMIMEv3] "S/MIME version 3".

This group of documents represents S/MIME version 3. This set of documents are [RFC2630], [RFC2631], [RFC2632], [RFC2633], [RFC2634], and [RFC5035].

[SMIMEv3.1] "S/MIME version 3.1".

This group of documents represents S/MIME version 3.1. This set of documents are [RFC2634], [RFC3850], [RFC3851], [RFC3852], and [RFC5035].

[SP800-57] National Institute of Standards and Technology (NIST), "Special Publication 800-57: Recommendation for Key Management", August 2005.

[X.500] "ITU-T Recommendation X.500 (1997) | ISO/IEC 9594- 1:1997, Information technology - Open Systems Interconnection - The Directory: Overview of concepts, models and services.".

Appendix A. Historic Considerations

A.1. Signature Algorithms and Key Sizes

There are a number of problems with validating certificates on sufficiently historic messages. For this reason it is strongly suggested that UAs treat these certificates differently from those on current messages. These problems include:

- CAs are not required to keep certificates on a CRL beyond one update after a certificate has expired. This means that unless CRLs are cached as part of the message it is not always possible to check if a certificate has been revoked. The same problems exist with OCSP responses as they may be based on a CRL rather than on the certificate database.

- RSA and DSA keys of less than 2048 bits are now considered by many experts to be cryptographically insecure (due to advances in computing power). Such keys were previously considered secure, so processing of historic certificates will often result in the use of weak keys. Implementations that wish to support previous versions of S/MIME or process old messages need to consider the security risks that result from smaller key sizes (e.g., spoofed messages) versus the costs of denial of service.

  [SMIMEv3.1] set the lower limit on suggested key sizes for creating and validation at 1024 bits. Prior to that the lower bound on key sizes was 512 bits.

- Hash functions used to validate signatures on historic messages may no longer be considered to be secure (see below). While there are not currently any known practical pre-image or second pre-image attacks against MD5 or SHA-1, the fact they are no longer considered to be collision resistant implies that the security level of any signature that is created with that these hash algorithms should also be considered as suspect.

The following algorithms have been called out for some level of support by previous S/MIME specifications:

- RSA with MD5 was dropped in [SMIMEv4.0]. MD5 is no longer considered to be secure as it is no longer collision-resistant. Details can be found in [RFC6151].

- RSA and DSA with SHA-1 were dropped in [SMIMEv4.0]. SHA-1 is no longer considered to be secure as it is no longer collision-resistant. The IETF statement on SHA-1 can be found in [RFC6194] but it is out-of-date relative to the most recent advances.

- DSA with SHA-256 support was dropped in [SMIMEv4.0]. DSA was dropped as part of a general movement from finite fields to elliptic curves. Issues have come up dealing with non-deterministic generation of the parameter 'k' (see [RFC6979]).

For 512-bit RSA with SHA-1 see [RFC3279] and [FIPS186-2] without Change Notice 1, for 512-bit RSA with SHA-256 see [RFC4055] and [FIPS186-2] without Change Notice 1.

For 512-bit DSA with SHA-1 see [RFC3279] and [FIPS186-2] without Change Notice 1, for 512-bit DSA with SHA-256 see [RFC5758] and [FIPS186-2] without Change Notice 1, for 1024-bit DSA with SHA-1 see [RFC3279] and [FIPS186-2] with Change Notice 1, for 1024-bit through 3072 DSA with SHA-256 see [RFC5758] and [FIPS186-3]. In either case, the first reference provides the signature algorithm's object identifier and the second provides the signature algorithm's definition.

Appendix B. Moving S/MIME v2 Certificate Handling to Historic Status

The S/MIME v3 [SMIMEv3], v3.1 [SMIMEv3.1], v3.2 [SMIMEv3.2], and v4.0 (this document) are backward compatible with the S/MIME v2 Certificate Handling Specification [SMIMEv2], with the exception of the algorithms (dropped RC2/40 requirement and added DSA and RSASSA-PSS requirements). Therefore, RFC 2312 [SMIMEv2] was moved to Historic status.

Appendix C. Acknowledgments

Many thanks go out to the other authors of the S/MIME v2 RFC: Steve Dusse, Paul Hoffman, and Jeff Weinstein. Without v2, there wouldn't be a v3, v3.1, v3.2 or v4.0.

A number of the members of the S/MIME Working Group have also worked very hard and contributed to this document. Any list of people is doomed to omission, and for that I apologize. In alphabetical order, the following people stand out in my mind because they made direct contributions to this document.

Bill Flanigan, Trevor Freeman, Elliott Ginsburg, Alfred Hoenes, Paul Hoffman, Russ Housley, David P. Kemp, Michael Myers, John Pawling, and Denis Pinkas.

The version 4 update to the S/MIME documents was done under the auspices of the LAMPS Working Group.

Authors' Addresses

Jim Schaad
August Cellars

Email: ietf@augustcellars.com

Blake Ramsdell
Brute Squad Labs, Inc.

Email: blaker@gmail.com

Sean Turner
sn3rd

Email: sean@sn3rd.com

LAMPS
Internet-Draft
Obsoletes: 5751 (if approved)
Intended status: Standards Track
Expires: March 8, 2019

J. Schaad
August Cellars
B. Ramsdell
Brute Squad Labs, Inc.
S. Turner
sn3rd

September 4, 2018

Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification draft-ietf-lamps-rfc5751-bis-12

Abstract

This document defines Secure/Multipurpose Internet Mail Extensions (S/MIME) version 4.0. S/MIME provides a consistent way to send and receive secure MIME data. Digital signatures provide authentication, message integrity, and non-repudiation with proof of origin. Encryption provides data confidentiality. Compression can be used to reduce data size. This document obsoletes RFC 5751.

Contributing to this document

The source for this draft is being maintained in GitHub. Suggested changes should be submitted as pull requests at <https://github.com/lamps-wg/smime>. Instructions are on that page as well. Editorial changes can be managed in GitHub, but any substantial issues need to be discussed on the LAMPS mailing list.

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 https://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 March 8, 2019.

Copyright Notice

Copyright (c) 2018 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 (https://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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction

S/MIME (Secure/Multipurpose Internet Mail Extensions) provides a consistent way to send and receive secure MIME data. Based on the popular Internet MIME standard, S/MIME provides the following cryptographic security services for electronic messaging applications: authentication, message integrity and non-repudiation of origin (using digital signatures), and data confidentiality (using encryption). As a supplementary service, S/MIME provides message compression.

S/MIME can be used by traditional mail user agents (MUAs) to add cryptographic security services to mail that is sent, and to interpret cryptographic security services in mail that is received. However, S/MIME is not restricted to mail; it can be used with any transport mechanism that transports MIME data, such as HTTP or SIP. As such, S/MIME takes advantage of the object-based features of MIME and allows secure messages to be exchanged in mixed-transport systems.

Further, S/MIME can be used in automated message transfer agents that use cryptographic security services that do not require any human intervention, such as the signing of software-generated documents and the encryption of FAX messages sent over the Internet.

This document defines the version 4.0 of the S/MIME Message specification. As such this document obsoletes version 3.2 of the S/MIME Message specification [RFC5751].

1.1. Specification Overview

This document describes a protocol for adding cryptographic signature and encryption services to MIME data. The MIME standard [MIME-SPEC] provides a general structure for the content of Internet messages and allows extensions for new content-type-based applications.

This specification defines how to create a MIME body part that has been cryptographically enhanced according to the Cryptographic Message Syntax (CMS) [CMS], which is derived from PKCS #7 [RFC2315]. This specification also defines the application/pkcs7-mime media type that can be used to transport those body parts.

This document also discusses how to use the multipart/signed media type defined in [RFC1847] to transport S/MIME signed messages. multipart/signed is used in conjunction with the application/pkcs7-signature media type, which is used to transport a detached S/MIME signature.

In order to create S/MIME messages, an S/MIME agent MUST follow the specifications in this document, as well as the specifications listed in the Cryptographic Message Syntax document [CMS], [RFC3370], [RFC4056], [RFC3560], and [RFC5754].

Throughout this specification, there are requirements and recommendations made for how receiving agents handle incoming messages. There are separate requirements and recommendations for how sending agents create outgoing messages. In general, the best strategy is to "be liberal in what you receive and conservative in what you send". Most of the requirements are placed on the handling of incoming messages, while the recommendations are mostly on the creation of outgoing messages.

The separation for requirements on receiving agents and sending agents also derives from the likelihood that there will be S/MIME systems that involve software other than traditional Internet mail clients. S/MIME can be used with any system that transports MIME data. An automated process that sends an encrypted message might not be able to receive an encrypted message at all, for example. Thus, the requirements and recommendations for the two types of agents are listed separately when appropriate.

1.2. Definitions

For the purposes of this specification, the following definitions apply.

ASN.1: Abstract Syntax Notation One, as defined in ITU-T Recommendations X.680, X.681, X.682 and X.683 [ASN.1].

BER: Basic Encoding Rules for ASN.1, as defined in ITU-T Recommendation X.690 [X.690].

Certificate: A type that binds an entity’s name to a public key with a digital signature.

DER: Distinguished Encoding Rules for ASN.1, as defined in ITU-T Recommendation X.690 [X.690].

7-bit data: Text data with lines less than 998 characters long, where none of the characters have the 8th bit set, and there are no NULL characters. <CR> and <LF> occur only as part of a <CR><LF> end-of-line delimiter.

8-bit data: Text data with lines less than 998 characters, and where none of the characters are NULL characters. <CR> and <LF> occur only as part of a <CR><LF> end-of-line delimiter.

Binary data: Arbitrary data.

Transfer encoding: A reversible transformation made on data so 8-bit or binary data can be sent via a channel that only transmits 7-bit data.

Receiving agent: Software that interprets and processes S/MIME CMS objects, MIME body parts that contain CMS content types, or both.

Sending agent: Software that creates S/MIME CMS content types, MIME body parts that contain CMS content types, or both.

S/MIME agent: User software that is a receiving agent, a sending agent, or both.

Data Integrity Service: A security service that protects against unauthorized changes to data by ensuring that changes to the data are detectable. [RFC4949]

Data Confidentiality: The property that data is not disclosed to system entities unless they have been authorized to know the data. [RFC4949]

1.3. Conventions Used 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

We define the additional requirement levels:

SHOULD+ This term means the same as SHOULD. However, the authors expect that a requirement marked as SHOULD+ will be promoted at some future time to be a MUST.

SHOULD- This term means the same as SHOULD. However, the authors expect that a requirement marked as SHOULD- will be demoted to a MAY in a future version of this document.

MUST- This term means the same as MUST. However, the authors expect that this requirement will no longer be a MUST in a future document. Although its status will be determined at a later time, it is reasonable to expect that if a future revision of a document alters the status of a MUST-requirement, it will remain at least a SHOULD or a SHOULD-.

The term RSA in this document almost always refers to the PKCS#1 v1.5 RSA [RFC2313] signature or encryption algorithms even when not qualified as such. There are a couple of places where it refers to the general RSA cryptographic operation, these can be determined from the context where it is used.

1.4. Compatibility with Prior Practice of S/MIME

S/MIME version 4.0 agents ought to attempt to have the greatest interoperability possible with agents for prior versions of S/MIME. S/MIME version 2 is described in RFC 2311 through RFC 2315 inclusive [SMIMEv2], S/MIME version 3 is described in RFC 2630 through RFC 2634 inclusive and RFC 5035 [SMIMEv3], S/MIME version 3.1 is described in RFC 3850, RFC 3851, RFC 3852, RFC 2634, and RFC 5035 [SMIMEv3.1], and S/MIME version 3.2 is described in [SMIMEv3.2]. [RFC2311] also has historical information about the development of S/MIME.

1.5. Changes from S/MIME v3 to S/MIME v3.1

This section describes the changes made between S/MIME v3 and S/MIME v3.1.

The RSA public key algorithm was changed to a MUST implement. Key wrap algorithm and the Diffie-Hellman (DH) algorithm [RFC2631] changed to a SHOULD implement.

The AES symmetric encryption algorithm has been included as a SHOULD implement.

The RSA public key algorithm was changed to a MUST implement signature algorithm.

Ambiguous language about the use of "empty" SignedData messages to transmit certificates was clarified to reflect that transmission of Certificate Revocation Lists is also allowed.

The use of binary encoding for some MIME entities is now explicitly discussed.

Header protection through the use of the message/rfc822 media type has been added.

Use of the CompressedData CMS type is allowed, along with required media type and file extension additions.

1.6. Changes from S/MIME v3.1 to S/MIME v3.2

This section describes the changes made between S/MIME v3.1 and S/MIME v3.2.

Editorial changes, e.g., replaced "MIME type" with "media type", content-type with Content-Type.

Moved "Conventions Used in This Document" to Section 1.3. Added definitions for SHOULD+, SHOULD-, and MUST-.

Section 1.1 and Appendix A: Added references to RFCs for RSASSA-PSS, RSAES-OAEP, and SHA2 CMS algorithms. Added CMS Multiple Signers Clarification to CMS reference.

Section 1.2: Updated references to ASN.1 to X.680 and BER and DER to X.690.

Section 1.4: Added references to S/MIME MSG 3.1 RFCs.

Section 2.1 (digest algorithm): SHA-256 added as MUST, SHA-1 and MD5 made SHOULD-.

Section 2.2 (signature algorithms): RSA with SHA-256 added as MUST, and DSA with SHA-256 added as SHOULD+, RSA with SHA-1, DSA with SHA-1, and RSA with MD5 changed to SHOULD-, and RSASSA-PSS with SHA-256 added as SHOULD+. Also added note about what S/MIME v3.1 clients support.

Section 2.3 (key encryption): DH changed to SHOULD-, and RSAES-OAEP added as SHOULD+. Elaborated requirements for key wrap algorithm.

Section 2.5.1: Added requirement that receiving agents MUST support both GeneralizedTime and UTCTime.

Section 2.5.2: Replaced reference "sha1WithRSAEncryption" with "sha256WithRSAEncryption", "DES-3EDE-CBC" with "AES-128 CBC", and deleted the RC5 example.

Section 2.5.2.1: Deleted entire section (discussed deprecated RC2).

Section 2.7, 2.7.1, Appendix A: references to RC2/40 removed.

Section 2.7 (content encryption): AES-128 CBC added as MUST, AES-192 and AES-256 CBC SHOULD+, tripleDES now SHOULD-.

Section 2.7.1: Updated pointers from 2.7.2.1 through 2.7.2.4 to 2.7.1.1 to 2.7.1.2.

Section 3.1.1: Removed text about MIME character sets.

Section 3.2.2 and 3.6: Replaced "encrypted" with "enveloped". Update OID example to use AES-128 CBC oid.

Section 3.4.3.2: Replace "micalg" parameter for "SHA-1" with "sha-1".

Section 4: Updated reference to CERT v3.2.

Section 4.1: Updated RSA and DSA key size discussion. Moved last four sentences to security considerations. Updated reference to randomness requirements for security.

Section 5: Added IANA registration templates to update media type registry to point to this document as opposed to RFC 2311.

Section 6: Updated security considerations.

Section 7: Moved references from Appendix B to this section. Updated references. Added informational references to SMIMEv2, SMIMEv3, and SMIMEv3.1.

Appendix C: Added Appendix C to move S/MIME v2 to Historic status.

1.7. Changes for S/MIME v4.0

This section describes the changes made between S/MIME v3.2 and S/MIME v4.0.

- Add the use of AuthEnvelopedData, including defining and registering an smime-type value (Section 2.4.4 and Section 3.4).

- Update the content encryption algorithms (Section 2.7 and Section 2.7.1.2): Add AES-256 GCM, add ChaCha200-Poly1305, remove mention of AES-192 CBC, mark tripleDES as historic.

- Update the set of signature algorithms (Section 2.2): Add Edwards-curve DSA (EdDSA) and ECDSA, mark DSA as historic

- Update the set of digest algorithms (Section 2.1): Add SHA-512, mark SHA-1 as historic.

- Update the size of keys to be used for RSA encryption and RSA signing (Section 4).

- Create Appendix B which deals with considerations for dealing with historic email messages.

2. CMS Options

CMS allows for a wide variety of options in content, attributes, and algorithm support. This section puts forth a number of support requirements and recommendations in order to achieve a base level of interoperability among all S/MIME implementations. [RFC3370] and [RFC5754] provides additional details regarding the use of the cryptographic algorithms. [ESS] provides additional details regarding the use of additional attributes.

2.1. DigestAlgorithmIdentifier

The algorithms here are used for digesting the body of the message and are not the same as the digest algorithms used as part the signature algorithms. The result of this is placed in the message-digest attribute of the signed attributes. It is RECOMMENDED that the algorithm used for digesting the body of the message be of similar or greater strength than the signature algorithm.

Sending and Receiving agents:

- MUST support SHA-256.

- MUST support SHA-512.

[RFC5754] provides the details for using these algorithms with S/MIME.

2.2. SignatureAlgorithmIdentifier

There are different sets of requirements placed on receiving and sending agents. By having the different requirements, the maximum amount of interoperability is achieved as it allows for specialized protection of private key material but maximum signature validation.

Receiving agents:

- MUST support ECDSA with curve P-256 and SHA-256.
- MUST support EdDSA with curve 25519 using Pure EdDSA mode [I-D.ietf-curdle-cms-eddsa-signatures].
- MUST- support RSA PKCS#1 v1.5 with SHA-256.
- SHOULD support RSASSA-PSS with SHA-256.

Sending agents:

- MUST support at least one of the following algorithms: ECDSA with curve P-256 and SHA-256, or EdDSA with curve 25519 using PureEdDSA mode.
- MUST- support RSA PKCS#1 v1.5 with SHA-256.
- SHOULD support RSASSA-PSS with SHA-256.

See Section 4.1 for information on key size and algorithm references.

2.3. KeyEncryptionAlgorithmIdentifier

Receiving and sending agents:

- MUST support ECDH ephemeral-static mode for P-256, as specified in [RFC5753].
- MUST support ECDH ephemeral-static mode for X25519 using HKDF-256 for the KDF, as specified in [I-D.ietf-curdle-cms-ecdh-new-curves].
- MUST- support RSA Encryption, as specified in [RFC3370].
- SHOULD+ support RSAES-OAEP, as specified in [RFC3560].

When ECDH ephemeral-static is used, a key wrap algorithm is also specified in the KeyEncryptionAlgorithmIdentifier [RFC5652]. The underlying encryption functions for the key wrap and content

encryption algorithm ([RFC3370] and [RFC3565]) and the key sizes for the two algorithms MUST be the same (e.g., AES-128 key wrap algorithm with AES-128 content encryption algorithm). As both 128 and 256 bit AES modes are mandatory-to-implement as content encryption algorithms (Section 2.7), both the AES-128 and AES-256 key wrap algorithms MUST be supported when ECDH ephemeral-static is used. Recipients MAY enforce this, but MUST use the weaker of the two as part of any cryptographic strength computation it might do.

Appendix B provides information on algorithms support in older versions of S/MIME.

2.4. General Syntax

There are several CMS content types. Of these, only the Data, SignedData, EnvelopedData, AuthEnvelopedData, and CompressedData content types are currently used for S/MIME.

2.4.1. Data Content Type

Sending agents MUST use the id-data content type identifier to identify the "inner" MIME message content. For example, when applying a digital signature to MIME data, the CMS SignedData encapContentInfo eContentType MUST include the id-data object identifier and the media type MUST be stored in the SignedData encapContentInfo eContent OCTET STRING (unless the sending agent is using multipart/signed, in which case the eContent is absent, per Section 3.5.3 of this document). As another example, when applying encryption to MIME data, the CMS EnvelopedData encryptedContentInfo contentType MUST include the id-data object identifier and the encrypted MIME content MUST be stored in the EnvelopedData encryptedContentInfo encryptedContent OCTET STRING.

2.4.2. SignedData Content Type

Sending agents MUST use the SignedData content type to apply a digital signature to a message or, in a degenerate case where there is no signature information, to convey certificates. Applying a signature to a message provides authentication, message integrity, and non-repudiation of origin.

2.4.3. EnvelopedData Content Type

This content type is used to apply data confidentiality to a message. In order to distribute the symmetric key, a sender needs to have access to a public key for each intended message recipient to use this service.

2.4.4. AuthEnvelopedData Content Type

This content type is used to apply data confidentiality and message integrity to a message. This content type does not provide authentication or non-repudiation. In order to distribute the symmetric key, a sender needs to have access to a public key for each intended message recipient to use this service.

2.4.5. CompressedData Content Type

This content type is used to apply data compression to a message. This content type does not provide authentication, message integrity, non-repudiation, or data confidentiality, and is only used to reduce the message's size.

See Section 3.7 for further guidance on the use of this type in conjunction with other CMS types.

2.5. Attributes and the SignerInfo Type

The SignerInfo type allows the inclusion of unsigned and signed attributes along with a signature. These attributes can be required for processing of message (i.e. Message Digest), information the signer supplied (i.e. SMIME Capabilities) that should be processed, or attributes which are not relevant in the current situation (i.e. mlExpansionList [RFC2634] for mail viewers).

Receiving agents MUST be able to handle zero or one instance of each of the signed attributes listed here. Sending agents SHOULD generate one instance of each of the following signed attributes in each S/MIME message:

- Signing Time (Section 2.5.1 in this document)
- SMIME Capabilities (Section 2.5.2 in this document)
- Encryption Key Preference (Section 2.5.3 in this document)
- Message Digest (Section 11.2 in [RFC5652])
- Content Type (Section 11.1 in [RFC5652])

Further, receiving agents SHOULD be able to handle zero or one instance of the signingCertificate and signingCertificatev2 signed attributes, as defined in Section 5 of RFC 2634 [ESS] and Section 3 of RFC 5035 [ESS].

Sending agents SHOULD generate one instance of the signingCertificate or signingCertificatev2 signed attribute in each SignerInfo structure.

Additional attributes and values for these attributes might be defined in the future. Receiving agents SHOULD handle attributes or values that they do not recognize in a graceful manner.

Interactive sending agents that include signed attributes that are not listed here SHOULD display those attributes to the user, so that the user is aware of all of the data being signed.

2.5.1. Signing Time Attribute

The signing-time attribute is used to convey the time that a message was signed. The time of signing will most likely be created by a signer and therefore is only as trustworthy as that signer.

Sending agents MUST encode signing time through the year 2049 as UTCTime; signing times in 2050 or later MUST be encoded as GeneralizedTime. When the UTCTime CHOICE is used, S/MIME agents MUST interpret the year field (YY) as follows:

> If YY is greater than or equal to 50, the year is interpreted as 19YY; if YY is less than 50, the year is interpreted as 20YY.

Receiving agents MUST be able to process signing-time attributes that are encoded in either UTCTime or GeneralizedTime.

2.5.2. SMIME Capabilities Attribute

The SMIMECapabilities attribute includes signature algorithms (such as "sha256WithRSAEncryption"), symmetric algorithms (such as "AES-128 CBC"), authenticated symmetric algorithms (such as "AES-128 GCM") and key encipherment algorithms (such as "rsaEncryption"). The presence of an algorithm based SMIME Capability attribute in this sequence implies that the sender can deal with the algorithm as well as understand the ASN.1 structures associated with that algorithm. There are also several identifiers that indicate support for other optional features such as binary encoding and compression. The SMIMECapabilities were designed to be flexible and extensible so that, in the future, a means of identifying other capabilities and preferences such as certificates can be added in a way that will not cause current clients to break.

If present, the SMIMECapabilities attribute MUST be a SignedAttribute. CMS defines SignedAttributes as a SET OF Attribute. The SignedAttributes in a signerInfo MUST include a single instance

of the SMIMECapabilities attribute. CMS defines the ASN.1 syntax for Attribute to include attrValues SET OF AttributeValue. A SMIMECapabilities attribute MUST only include a single instance of AttributeValue. If a signature is detected as violating these requirements, the signature SHOULD be treated as failing.

The semantics of the SMIMECapabilities attribute specify a partial list as to what the client announcing the SMIMECapabilities can support. A client does not have to list every capability it supports, and need not list all its capabilities so that the capabilities list doesn't get too long. In an SMIMECapabilities attribute, the object identifiers (OIDs) are listed in order of their preference, but SHOULD be separated logically along the lines of their categories (signature algorithms, symmetric algorithms, key encipherment algorithms, etc.).

The structure of the SMIMECapabilities attribute is to facilitate simple table lookups and binary comparisons in order to determine matches. For instance, the encoding for the SMIMECapability for sha256WithRSAEncryption includes rather than omits the NULL parameter. Because of the requirement for identical encoding, individuals documenting algorithms to be used in the SMIMECapabilities attribute SHOULD explicitly document the correct byte sequence for the common cases.

For any capability, the associated parameters for the OID MUST specify all of the parameters necessary to differentiate between two instances of the same algorithm.

The OIDs that correspond to algorithms SHOULD use the same OID as the actual algorithm, except in the case where the algorithm usage is ambiguous from the OID. For instance, in an earlier specification, rsaEncryption was ambiguous because it could refer to either a signature algorithm or a key encipherment algorithm. In the event that an OID is ambiguous, it needs to be arbitrated by the maintainer of the registered SMIMECapabilities list as to which type of algorithm will use the OID, and a new OID MUST be allocated under the smimeCapabilities OID to satisfy the other use of the OID.

The registered SMIMECapabilities list specifies the parameters for OIDs that need them, most notably key lengths in the case of variable-length symmetric ciphers. In the event that there are no differentiating parameters for a particular OID, the parameters MUST be omitted, and MUST NOT be encoded as NULL. Additional values for the SMIMECapabilities attribute might be defined in the future. Receiving agents MUST handle a SMIMECapabilities object that has values that it does not recognize in a graceful manner.

Section 2.7.1 explains a strategy for caching capabilities.

2.5.3. Encryption Key Preference Attribute

The encryption key preference attribute allows the signer to unambiguously describe which of the signer's certificates has the signer's preferred encryption key. This attribute is designed to enhance behavior for interoperating with those clients that use separate keys for encryption and signing. This attribute is used to convey to anyone viewing the attribute which of the listed certificates is appropriate for encrypting a session key for future encrypted messages.

If present, the SMIMEEncryptionKeyPreference attribute MUST be a SignedAttribute. CMS defines SignedAttributes as a SET OF Attribute. The SignedAttributes in a signerInfo MUST include a single instance of the SMIMEEncryptionKeyPreference attribute. CMS defines the ASN.1 syntax for Attribute to include attrValues SET OF AttributeValue. A SMIMEEncryptionKeyPreference attribute MUST only include a single instance of AttributeValue. If a signature is detected to violate these requirements, the signature SHOULD be treated as failing.

The sending agent SHOULD include the referenced certificate in the set of certificates included in the signed message if this attribute is used. The certificate MAY be omitted if it has been previously made available to the receiving agent. Sending agents SHOULD use this attribute if the commonly used or preferred encryption certificate is not the same as the certificate used to sign the message.

Receiving agents SHOULD store the preference data if the signature on the message is valid and the signing time is greater than the currently stored value. (As with the SMIMECapabilities, the clock skew SHOULD be checked and the data not used if the skew is too great.) Receiving agents SHOULD respect the sender's encryption key preference attribute if possible. This, however, represents only a preference and the receiving agent can use any certificate in replying to the sender that is valid.

Section 2.7.1 explains a strategy for caching preference data.

2.5.3.1. Selection of Recipient Key Management Certificate

In order to determine the key management certificate to be used when sending a future CMS EnvelopedData message for a particular recipient, the following steps SHOULD be followed:

- If an SMIMEEncryptionKeyPreference attribute is found in a SignedData object received from the desired recipient, this identifies the X.509 certificate that SHOULD be used as the X.509 key management certificate for the recipient.

- If an SMIMEEncryptionKeyPreference attribute is not found in a SignedData object received from the desired recipient, the set of X.509 certificates SHOULD be searched for a X.509 certificate with the same subject name as the signer of a X.509 certificate that can be used for key management.

- Or use some other method of determining the user's key management key. If a X.509 key management certificate is not found, then encryption cannot be done with the signer of the message. If multiple X.509 key management certificates are found, the S/MIME agent can make an arbitrary choice between them.

2.6. SignerIdentifier SignerInfo Type

S/MIME v4.0 implementations MUST support both issuerAndSerialNumber and subjectKeyIdentifier. Messages that use the subjectKeyIdentifier choice cannot be read by S/MIME v2 clients.

It is important to understand that some certificates use a value for subjectKeyIdentifier that is not suitable for uniquely identifying a certificate. Implementations MUST be prepared for multiple certificates for potentially different entities to have the same value for subjectKeyIdentifier, and MUST be prepared to try each matching certificate during signature verification before indicating an error condition.

2.7. ContentEncryptionAlgorithmIdentifier

Sending and receiving agents:

- MUST support encryption and decryption with AES-128 GCM and AES-256 GCM [RFC5084].

- MUST- support encryption and decryption with AES-128 CBC [RFC3565].

- SHOULD+ support encryption and decryption with ChaCha20-Poly1305 [RFC7905].

2.7.1. Deciding Which Encryption Method to Use

When a sending agent creates an encrypted message, it has to decide which type of encryption to use. The decision process involves using information garnered from the capabilities lists included in messages received from the recipient, as well as out-of-band information such as private agreements, user preferences, legal restrictions, and so on.

Section 2.5.2 defines a method by which a sending agent can optionally announce, among other things, its decrypting capabilities in its order of preference. The following method for processing and remembering the encryption capabilities attribute in incoming signed messages SHOULD be used.

- If the receiving agent has not yet created a list of capabilities for the sender's public key, then, after verifying the signature on the incoming message and checking the timestamp, the receiving agent SHOULD create a new list containing at least the signing time and the symmetric capabilities.

- If such a list already exists, the receiving agent SHOULD verify that the signing time in the incoming message is greater than the signing time stored in the list and that the signature is valid. If so, the receiving agent SHOULD update both the signing time and capabilities in the list. Values of the signing time that lie far in the future (that is, a greater discrepancy than any reasonable clock skew), or a capabilities list in messages whose signature could not be verified, MUST NOT be accepted.

The list of capabilities SHOULD be stored for future use in creating messages.

Before sending a message, the sending agent MUST decide whether it is willing to use weak encryption for the particular data in the message. If the sending agent decides that weak encryption is unacceptable for this data, then the sending agent MUST NOT use a weak algorithm. The decision to use or not use weak encryption overrides any other decision in this section about which encryption algorithm to use.

Section 2.7.1.1 and Section 2.7.1.2 describe the decisions a sending agent SHOULD use in deciding which type of encryption will be applied to a message. These rules are ordered, so the sending agent SHOULD make its decision in the order given.

2.7.1.1. Rule 1: Known Capabilities

If the sending agent has received a set of capabilities from the recipient for the message the agent is about to encrypt, then the sending agent SHOULD use that information by selecting the first capability in the list (that is, the capability most preferred by the intended recipient) that the sending agent knows how to encrypt. The sending agent SHOULD use one of the capabilities in the list if the agent reasonably expects the recipient to be able to decrypt the message.

2.7.1.2. Rule 2: Unknown Capabilities, Unknown Version of S/MIME

If the following two conditions are met:

- the sending agent has no knowledge of the encryption capabilities of the recipient, and

- the sending agent has no knowledge of the version of S/MIME of the recipient,

then the sending agent SHOULD use AES-256 GCM because it is a stronger algorithm and is required by S/MIME v4.0. If the sending agent chooses not to use AES-256 GCM in this step, given the presumption is that a client implementing AES-GCM would do both AES-256 and AES-128, it SHOULD use AES-128 CBC.

2.7.2. Choosing Weak Encryption

Algorithms such as RC2 are considered to be weak encryption algorithms. Algorithms such as TripleDES are not state of the art and are considered to be weaker algorithms than AES. A sending agent that is controlled by a human SHOULD allow a human sender to determine the risks of sending data using a weaker encryption algorithm before sending the data, and possibly allow the human to use a stronger encryption algorithm such as AES GCM or AES CBC even if there is a possibility that the recipient will not be able to process that algorithm.

2.7.3. Multiple Recipients

If a sending agent is composing an encrypted message to a group of recipients where the encryption capabilities of some of the recipients do not overlap, the sending agent is forced to send more than one message. Please note that if the sending agent chooses to send a message encrypted with a strong algorithm, and then send the same message encrypted with a weak algorithm, someone watching the

communications channel could learn the contents of the strongly encrypted message simply by decrypting the weakly encrypted message.

3. Creating S/MIME Messages

This section describes the S/MIME message formats and how they are created. S/MIME messages are a combination of MIME bodies and CMS content types. Several media types as well as several CMS content types are used. The data to be secured is always a canonical MIME entity. The MIME entity and other data, such as certificates and algorithm identifiers, are given to CMS processing facilities that produce a CMS object. Finally, the CMS object is wrapped in MIME. The Enhanced Security Services for S/MIME [ESS] document provides descriptions of how nested, secured S/MIME messages are formatted. ESS provides a description of how a triple-wrapped S/MIME message is formatted using multipart/signed and application/pkcs7-mime for the signatures.

S/MIME provides one format for enveloped-only data, several formats for signed-only data, and several formats for signed and enveloped data. Several formats are required to accommodate several environments, in particular for signed messages. The criteria for choosing among these formats are also described.

The reader of this section is expected to understand MIME as described in [MIME-SPEC] and [RFC1847].

3.1. Preparing the MIME Entity for Signing, Enveloping, or Compressing

S/MIME is used to secure MIME entities. A MIME message is composed of a MIME header and a MIME body. The body can consist of a single part or of multiple parts. Any of these parts is designated as a MIME message part. A MIME entity can be a sub-part, sub-parts of a MIME message, or the whole MIME message with all of its sub-parts. A MIME entity that is the whole message includes only the MIME message headers and MIME body, and does not include the RFC-822 header. Note that S/MIME can also be used to secure MIME entities used in applications other than Internet mail. If protection of the RFC-822 header is required, the use of the message/rfc822 media type is explained later in this section.

The MIME entity that is secured and described in this section can be thought of as the "inside" MIME entity. That is, it is the "innermost" object in what is possibly a larger MIME message. Processing "outside" MIME entities into CMS content types is described in Section 3.2, Section 3.5, and elsewhere.

The procedure for preparing a MIME entity is given in [MIME-SPEC]. The same procedure is used here with some additional restrictions when signing. The description of the procedures from [MIME-SPEC] is repeated here, but it is suggested that the reader refer to that document for the exact procedure. This section also describes additional requirements.

A single procedure is used for creating MIME entities that are to have any combination of signing, enveloping, and compressing applied. Some additional steps are recommended to defend against known corruptions that can occur during mail transport that are of particular importance for clear-signing using the multipart/signed format. It is recommended that these additional steps be performed on enveloped messages, or signed and enveloped messages, so that the message can be forwarded to any environment without modification.

These steps are descriptive rather than prescriptive. The implementer is free to use any procedure as long as the result is the same.

Step 1. The MIME entity is prepared according to the local conventions.

Step 2. The leaf parts of the MIME entity are converted to canonical form.

Step 3. Appropriate transfer encoding is applied to the leaves of the MIME entity.

When an S/MIME message is received, the security services on the message are processed, and the result is the MIME entity. That MIME entity is typically passed to a MIME-capable user agent where it is further decoded and presented to the user or receiving application.

In order to protect outer, non-content-related message header fields (for instance, the "Subject", "To", "From", and "Cc" fields), the sending client MAY wrap a full MIME message in a message/rfc822 wrapper in order to apply S/MIME security services to these header fields. It is up to the receiving client to decide how to present this "inner" header along with the unprotected "outer" header. Given the security difference between headers, it is RECOMMENDED that client provide a distinction between header fields depending on where they are located.

When an S/MIME message is received, if the top-level protected MIME entity has a Content-Type of message/rfc822, it can be assumed that the intent was to provide header protection. This entity SHOULD be

presented as the top-level message, taking into account header merging issues as previously discussed.

3.1.1. Canonicalization

Each MIME entity MUST be converted to a canonical form that is uniquely and unambiguously representable in the environment where the signature is created and the environment where the signature will be verified. MIME entities MUST be canonicalized for enveloping and compressing as well as signing.

The exact details of canonicalization depend on the actual media type and subtype of an entity, and are not described here. Instead, the standard for the particular media type SHOULD be consulted. For example, canonicalization of type text/plain is different from canonicalization of audio/basic. Other than text types, most types have only one representation regardless of computing platform or environment that can be considered their canonical representation. In general, canonicalization will be performed by the non-security part of the sending agent rather than the S/MIME implementation.

The most common and important canonicalization is for text, which is often represented differently in different environments. MIME entities of major type "text" MUST have both their line endings and character set canonicalized. The line ending MUST be the pair of characters <CR><LF>, and the charset SHOULD be a registered charset [CHARSETS]. The details of the canonicalization are specified in [MIME-SPEC].

Note that some charsets such as ISO-2022 have multiple representations for the same characters. When preparing such text for signing, the canonical representation specified for the charset MUST be used.

3.1.2. Transfer Encoding

When generating any of the secured MIME entities below, except the signing using the multipart/signed format, no transfer encoding is required at all. S/MIME implementations MUST be able to deal with binary MIME objects. If no Content-Transfer-Encoding header field is present, the transfer encoding is presumed to be 7BIT.

As a rule, S/MIME implementations SHOULD use transfer encoding described in Section 3.1.3 for all MIME entities they secure. The reason for securing only 7-bit MIME entities, even for enveloped data that is not exposed to the transport, is that it allows the MIME entity to be handled in any environment without changing it. For example, a trusted gateway might remove the envelope, but not the

signature, of a message, and then forward the signed message on to the end recipient so that they can verify the signatures directly. If the transport internal to the site is not 8-bit clean, such as on a wide-area network with a single mail gateway, verifying the signature will not be possible unless the original MIME entity was only 7-bit data.

In the case where S/MIME implementations can determine that all intended recipients are capable of handling inner (all but the outermost) binary MIME objects, implementations SHOULD use binary encoding as opposed to a 7-bit-safe transfer encoding for the inner entities. The use of a 7-bit-safe encoding (such as base64) unnecessarily expands the message size. Implementations MAY determine that recipient implementations are capable of handling inner binary MIME entities either by interpreting the id-cap-preferBinaryInside SMIMECapabilities attribute, by prior agreement, or by other means.

If one or more intended recipients are unable to handle inner binary MIME objects, or if this capability is unknown for any of the intended recipients, S/MIME implementations SHOULD use transfer encoding described in Section 3.1.3 for all MIME entities they secure.

3.1.3. Transfer Encoding for Signing Using multipart/signed

If a multipart/signed entity is ever to be transmitted over the standard Internet SMTP infrastructure or other transport that is constrained to 7-bit text, it MUST have transfer encoding applied so that it is represented as 7-bit text. MIME entities that are 7-bit data already need no transfer encoding. Entities such as 8-bit text and binary data can be encoded with quoted-printable or base-64 transfer encoding.

The primary reason for the 7-bit requirement is that the Internet mail transport infrastructure cannot guarantee transport of 8-bit or binary data. Even though many segments of the transport infrastructure now handle 8-bit and even binary data, it is sometimes not possible to know whether the transport path is 8-bit clean. If a mail message with 8-bit data were to encounter a message transfer agent that cannot transmit 8-bit or binary data, the agent has three options, none of which are acceptable for a clear-signed message:

- The agent could change the transfer encoding; this would invalidate the signature.

- The agent could transmit the data anyway, which would most likely result in the 8th bit being corrupted; this too would invalidate the signature.

- The agent could return the message to the sender.

[RFC1847] prohibits an agent from changing the transfer encoding of the first part of a multipart/signed message. If a compliant agent that cannot transmit 8-bit or binary data encountered a multipart/signed message with 8-bit or binary data in the first part, it would have to return the message to the sender as undeliverable.

3.1.4. Sample Canonical MIME Entity

This example shows a multipart/mixed message with full transfer encoding. This message contains a text part and an attachment. The sample message text includes characters that are not ASCII and thus need to be transfer encoded. Though not shown here, the end of each line is <CR><LF>. The line ending of the MIME headers, the text, and the transfer encoded parts, all MUST be <CR><LF>.

Note that this example is not of an S/MIME message.

```
Content-Type: multipart/mixed; boundary=bar

--bar
Content-Type: text/plain; charset=iso-8859-1
Content-Transfer-Encoding: quoted-printable

=A1Hola Michael!

How do you like the new S/MIME specification?

It’s generally a good idea to encode lines that begin with
From=20because some mail transport agents will insert a greater-
than (>) sign, thus invalidating the signature.

Also, in some cases it might be desirable to encode any =20
trailing whitespace that occurs on lines in order to ensure =20
that the message signature is not invalidated when passing =20
a gateway that modifies such whitespace (like BITNET). =20

--bar
Content-Type: image/jpeg
Content-Transfer-Encoding: base64

iQCVAwUBMJrRF2N9oWBghPDJAQE9UQQAtl7LuRVndBjrk4EqYBIb3h5QXIX/LC//
jJV5bNvkZIGPIcEmI5iFd9boEgvpirHtIREEqLQRkYNoBActFBZmh9GC3C041WGq
uMbrbxc+nIs1TIKlA08rVi9ig/2Yh7LFrK5Ein57U/W72vgSxLhe/zhdfolT9Brn
HOxEa44b+EI=

--bar--
```

3.2. The application/pkcs7-mime Media Type

The application/pkcs7-mime media type is used to carry CMS content types including EnvelopedData, SignedData, and CompressedData. The details of constructing these entities are described in subsequent sections. This section describes the general characteristics of the application/pkcs7-mime media type.

The carried CMS object always contains a MIME entity that is prepared as described in Section 3.1 if the eContentType is id-data. Other contents MAY be carried when the eContentType contains different values. See [ESS] for an example of this with signed receipts.

Since CMS content types are binary data, in most cases base-64 transfer encoding is appropriate, in particular, when used with SMTP transport. The transfer encoding used depends on the transport through which the object is to be sent, and is not a characteristic of the media type.

Note that this discussion refers to the transfer encoding of the CMS object or "outside" MIME entity. It is completely distinct from, and unrelated to, the transfer encoding of the MIME entity secured by the CMS object, the "inside" object, which is described in Section 3.1.

Because there are several types of application/pkcs7-mime objects, a sending agent SHOULD do as much as possible to help a receiving agent know about the contents of the object without forcing the receiving agent to decode the ASN.1 for the object. The Content-Type header field of all application/pkcs7-mime objects SHOULD include the optional "smime-type" parameter, as described in the following sections.

3.2.1. The name and filename Parameters

For the application/pkcs7-mime, sending agents SHOULD emit the optional "name" parameter to the Content-Type field for compatibility with older systems. Sending agents SHOULD also emit the optional Content-Disposition field [RFC2183] with the "filename" parameter. If a sending agent emits the above parameters, the value of the parameters SHOULD be a file name with the appropriate extension:

Media Type	File Extension
application/pkcs7-mime (SignedData, EnvelopedData, AuthEnvelopedData)	.p7m
application/pkcs7-mime (degenerate SignedData certificate management message)	.p7c
application/pkcs7-mime (CompressedData)	.p7z
application/pkcs7-signature (SignedData)	.p7s

In addition, the file name SHOULD be limited to eight characters followed by a three-letter extension. The eight-character filename base can be any distinct name; the use of the filename base "smime" SHOULD be used to indicate that the MIME entity is associated with S/MIME.

Including a file name serves two purposes. It facilitates easier use of S/MIME objects as files on disk. It also can convey type information across gateways. When a MIME entity of type application/pkcs7-mime (for example) arrives at a gateway that has no special knowledge of S/MIME, it will default the entity’s media type to application/octet-stream and treat it as a generic attachment, thus losing the type information. However, the suggested filename for an attachment is often carried across a gateway. This often allows the receiving systems to determine the appropriate application to hand the attachment off to, in this case, a stand-alone S/MIME processing application. Note that this mechanism is provided as a

convenience for implementations in certain environments. A proper S/MIME implementation MUST use the media types and MUST NOT rely on the file extensions.

3.2.2. The smime-type Parameter

The application/pkcs7-mime content type defines the optional "smime-type" parameter. The intent of this parameter is to convey details about the security applied (signed or enveloped) along with information about the contained content. This specification defines the following smime-types.

Name	CMS Type	Inner Content
enveloped-data	EnvelopedData	id-data
signed-data	SignedData	id-data
certs-only	SignedData	id-data
compressed-data	CompressedData	id-data
authEnveloped-data	AuthEnvelopedData	id-data

In order for consistency to be obtained with future specifications, the following guidelines SHOULD be followed when assigning a new smime-type parameter.

1. If both signing and encryption can be applied to the content, then three values for smime-type SHOULD be assigned "signed-*", "authEnv-*", and "enveloped-*". If one operation can be assigned, then this can be omitted. Thus, since "certs-only" can only be signed, "signed-" is omitted.

2. A common string for a content OID SHOULD be assigned. We use "data" for the id-data content OID when MIME is the inner content.

3. If no common string is assigned, then the common string of "OID.<oid>" is recommended (for example, "OID.2.16.840.1.101.3.4.1.2" would be AES-128 CBC).

It is explicitly intended that this field be a suitable hint for mail client applications to indicate whether a message is "signed", "authEnveloped" or "enveloped" without having to tunnel into the CMS payload.

A registry for additional smime-type parameter values has been defined in [RFC7114].

3.3. Creating an Enveloped-Only Message

This section describes the format for enveloping a MIME entity without signing it. It is important to note that sending enveloped but not signed messages does not provide for data integrity. The Enveloped-Only structure does not support authenticated symmetric algorithmm. Use the Authenticated Enveloped structure for these algorithms. Thus, it is possible to replace ciphertext in such a way that the processed message will still be valid, but the meaning can be altered.

Step 1. The MIME entity to be enveloped is prepared according to Section 3.1.

Step 2. The MIME entity and other required data is processed into a CMS object of type EnvelopedData. In addition to encrypting a copy of the content-encryption key for each recipient, a copy of the content-encryption key SHOULD be encrypted for the originator and included in the EnvelopedData (see [RFC5652], Section 6).

Step 3. The EnvelopedData object is wrapped in a CMS ContentInfo object.

Step 4. The ContentInfo object is inserted into an application/pkcs7-mime MIME entity.

The smime-type parameter for enveloped-only messages is "enveloped-data". The file extension for this type of message is ".p7m".

A sample message would be:

```
Content-Type: application/pkcs7-mime; name=smime.p7m;
        smime-type=enveloped-data
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

MIIBHgYJKoZIhvcNAQcDoIIBDzCCAQsCAQAxgcAwgb0CAQAwJjASMRAwDgYDVQQDEw
dDYXJsUlNBAhBGNGvHgABWvBHTbi7NXXHQMA0GCSqGSIb3DQEBAQUABIGAC3EN5nGI
iJi2lsGPcP2iJ97a4e8kbKQz36zg6Z2i0yx6zYC4mZ7mX7FBs3IWg+f6KgCLx3M1eC
bWx8+MDFbbpXadCDgO8/nUkUNYeNxJtuzubGgzoyEd8Ch4H/dd9gdzTd+taTEgS0ip
dSJuNnkVY4/M652jKKHRLFf02hosdR8wQwYJKoZIhvcNAQcBMBQGCCqGSIb3DQMHBA
gtaMXpRwZRNYAgDsiSf8Z9P43LrY4OxUk660cu1lXeCSFOSOpOJ7FuVyU=
```

3.4. Creating an Authenticated Enveloped-Only Message

This section describes the format for enveloping a MIME entity without signing it. Authenticated enveloped messages provide confidentiality and data integrity. It is important to note that sending authenticated enveloped messages does not provide for proof of origination when using S/MIME. It is possible for a third party to replace ciphertext in such a way that the processed message will still be valid, but the meaning can be altered. However this is substantially more difficult than it is for an enveloped-only message as the algorithm does provide a level of authentication. Any recipient for whom the message is encrypted can replace it without detection.

Step 1. The MIME entity to be enveloped is prepared according to Section 3.1.

Step 2. The MIME entity and other required data is processed into a CMS object of type AuthEnvelopedData. In addition to encrypting a copy of the content-encryption key for each recipient, a copy of the content-encryption key SHOULD be encrypted for the originator and included in the AuthEnvelopedData (see [RFC5083]).

Step 3. The AuthEnvelopedData object is wrapped in a CMS ContentInfo object.

Step 4. The ContentInfo object is inserted into an application/pkcs7-mime MIME entity.

The smime-type parameter for authenticated enveloped-only messages is "authEnveloped-data". The file extension for this type of message is ".p7m".

A sample message would be:

```
Content-Type: application/pkcs7-mime; smime-type=authEnveloped-data;
      name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

MIIDWQYLKoZIhvcNAQkQAREgggNIMIIDRAIBADGBvjCBuwIBADAmMBIxEDAO
BgNVBAMTB0NhcmxSU0ECEEY0a8eAAFa8EdNuLs1dcdAwCwYJKoZIhvcNAQEB
BIGAgyZJo0ERTxA4xdTri5P5tVMyh0RARepTUCORZvlUbcUlaI8IpJZH3/J1
Fv6MxTRS4O/K+ZcTlQmYeWLQvwdltQdOIP3mhpqXzTnOYhTK1IDtF2zx75Lg
vE+ilpcLIzXfJB4RCBPtBWaHAof4Wb+VMQvLkk9OolX4mRSH1LPktgAwggJq
BgkqhkiG9w0BBwEwGwYJYIZIAWUDBAEGMA4EDGPizioC9OHSsnNx4oCCAj7Y
Cb8rOy8+55106newEJohC/aDgWbJhrMKzSOwa7JraXOV3HXD3NvKbl665dRx
vmDwSCNaLCRU5q8/AxQx2SvnAbM+JKcEfC/VFdd4SiHNiUECAApLku2rMi5B
WrhW/FXmx9d+cjum2BRwB3wj0q1wajdB0/kVRbQwg697dnlYyUog4vpJERjr
7KAkawZx1RMHaM18wgZjUNpCBXFS3chQi9mTBp2i2Hf5iZ8OOtTx+rCQUmI6
Jhy03vdcPCCARBjn3v0d3upZYDZddMA41CB9fKnnWFjadV1KpYwv80tqsEfx
Vo0lJQ5VtJ8MHJiBpLVKadRIZ4iH2ULC0JtN5mXE1SrFKh7cqbJ4+7nqSRL3
oBTud3rX41DGshOjpqcYHT4sqYlgZkc6dp0g1+hF1p3cGmjHdpysV2NVSUev
ghHbvSqhIsXFzRSWKiZOigmlkv3R5LnjpYyP4brM62Jl7y0qborvV4dNMz7m
D+5YxSlH0KAe8z6TT3LHuQdN7QCkFoiUSCaNhpAFaakkGIpqcqLhpOK4lXxt
kptCG93eUwNCcTxtx6bXufPR5TUHohvZvfeqMp42kL37FJC/A8ZHoOxXy8+X
X5QYxCQNuofWlvnIWv0Nr8w65x6lgVjPYmd/cHwzQKBTBMXN6pBud/PZL5zF
tw3QHlQkBR+UflMWZKeN9L0KdQ27mQlCo5gQS85aifxoiiA2v9+0hxZw91rP
IW4D+GS7oMMoKj8ZNyCJJsyf5smRZ+WxeBoolb3+TiGcBBCsRnfe6noLZiFO
6Zeu2ZwE
```

3.5. Creating a Signed-Only Message

There are two formats for signed messages defined for S/MIME:

- application/pkcs7-mime with SignedData.

- multipart/signed.

In general, the multipart/signed form is preferred for sending, and receiving agents MUST be able to handle both.

3.5.1. Choosing a Format for Signed-Only Messages

There are no hard-and-fast rules as to when a particular signed-only format is chosen. It depends on the capabilities of all the receivers and the relative importance of receivers with S/MIME facilities being able to verify the signature versus the importance of receivers without S/MIME software being able to view the message.

Messages signed using the multipart/signed format can always be viewed by the receiver whether or not they have S/MIME software. They can also be viewed whether they are using a MIME-native user

agent or they have messages translated by a gateway. In this context, "be viewed" means the ability to process the message essentially as if it were not a signed message, including any other MIME structure the message might have.

Messages signed using the SignedData format cannot be viewed by a recipient unless they have S/MIME facilities. However, the SignedData format protects the message content from being changed by benign intermediate agents. Such agents might do line wrapping or content-transfer encoding changes that would break the signature.

3.5.2. Signing Using application/pkcs7-mime with SignedData

This signing format uses the application/pkcs7-mime media type. The steps to create this format are:

Step 1. The MIME entity is prepared according to Section 3.1.

Step 2. The MIME entity and other required data are processed into a CMS object of type SignedData.

Step 3. The SignedData object is wrapped in a CMS ContentInfo object.

Step 4. The ContentInfo object is inserted into an application/pkcs7-mime MIME entity.

The smime-type parameter for messages using application/pkcs7-mime with SignedData is "signed-data". The file extension for this type of message is ".p7m".

A sample message would be:

```
Content-Type: application/pkcs7-mime; smime-type=signed-data;
    name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

MIIDmQYJKoZIhvcNAQcCoIIDijCCA4YCAQExCTAHBgUrDgMCGjAtBgkqhkiG9w0BBw
GgIAQeDQpUaGlzIGlzIHNvbWUgc2FtcGxlIGNvbnRlbnQuoIIC4DCCAtwwggKboAMC
AQICAgDIMAkGByqGSM44BAMwEjEQMA4GA1UEAxMHQ2FybERTUzAeFw05OTA4MTcwMT
EwNDlaFw0zOTEyMzEyMzU5NTlaMBMxETAPBgNVBAMTCEFsaWNlRFNTMIIBtjCCASsG
ByqGSM44BAEwggEeAoGBAIGNze2D6gqeOT7CSCij5EeT3Q7XqA7sU8WrhAhP/5Thc0
h+DNbzREjR/p+vpKGJL+HZMMg23j+bv7dM3F9piuR10DcMkQiVm96nXvn89J8v3UOo
i1TxP7AHCEdNXYjDw7Wz41UIddU5dhDEeL3/nbCElzfy5FEbteQJllzzflvbAhUA4k
emGkVmuBPG2o+4NyErYov3k80CgYAmONAUiTKqOfs+bdlLWWpMdiM5BAI1XPLLGjDD
HlBd3ZtZ4s2qBT1YwHuiNrhuB699ikIlp/R1z0oIXks+kPht6pzJIYo7dhTpzi5dow
fNI4W4LzABfG1JiRGJNkS9+MiVSlNWteL5c+waYTYfEX/Cve3RUP+YdMLRgUpgObo2
OQOBhAACgYBc47ladRSWC6l63eM/qeysXty9txMRNKYWiSgRI9k0hmd1dRMSPUNbb+
VRv/qJ8qIbPiR9PQeNW2PIu0WloErjhdbOBoA/6CN+GvIkq1MauCcNHu8Iv2YUgFxi
rGX6FYvxuzTU0pY39mFHssQyhPB+QUD9RqdjTjPypeL08oPluKOBgTB/MAwGA1UdEw
EB/wQCMAAwDgYDVR0PAQH/BAQDAgbAMB8GA1UdIwQYMBaAFHBEPoIub4feStN14z0g
vEMrk/EfMB0GA1UdDgQWBBS+bKGz48H37UNwpM4TAeL945f+zTAfBgNVHREEGDAWgR
RBbGljZURTU0BleGFtcGxlLmNvbTAJBgcqhkjOOAQDAzAAMC0CFFUMpBkfQiuJcSIz
jYNqtT1na79FAhUAn2FTUlQLXLLd2ud2HeIQUltDXr0xYzBhAgEBMBgwEjEQMA4GA1
UEAxMHQ2FybERTUwICAMgwBwYFKw4DAhowCQYHKoZIzjgEAwQuMCwCFD1cSW6LIUFz
eXle3YI5SKSBer/sAhQmCq7s/CTFHOEjgASeUjbMpx5g6A==
```

3.5.3. Signing Using the multipart/signed Format

This format is a clear-signing format. Recipients without any S/MIME or CMS processing facilities are able to view the message. It makes use of the multipart/signed media type described in [RFC1847]. The multipart/signed media type has two parts. The first part contains the MIME entity that is signed; the second part contains the "detached signature" CMS SignedData object in which the encapContentInfo eContent field is absent.

3.5.3.1. The application/pkcs7-signature Media Type

This media type always contains a CMS ContentInfo containing a single CMS object of type SignedData. The SignedData encapContentInfo eContent field MUST be absent. The signerInfos field contains the signatures for the MIME entity.

The file extension for signed-only messages using application/pkcs7-signature is ".p7s".

3.5.3.2. Creating a multipart/signed Message

Step 1. The MIME entity to be signed is prepared according to Section 3.1, taking special care for clear-signing.

Step 2. The MIME entity is presented to CMS processing in order to obtain an object of type SignedData in which the encapContentInfo eContent field is absent.

Step 3. The MIME entity is inserted into the first part of a multipart/signed message with no processing other than that described in Section 3.1.

Step 4. Transfer encoding is applied to the "detached signature" CMS SignedData object, and it is inserted into a MIME entity of type application/pkcs7-signature.

Step 5. The MIME entity of the application/pkcs7-signature is inserted into the second part of the multipart/signed entity.

The multipart/signed Content-Type has two required parameters: the protocol parameter and the micalg parameter.

The protocol parameter MUST be "application/pkcs7-signature". Note that quotation marks are required around the protocol parameter because MIME requires that the "/" character in the parameter value MUST be quoted.

The micalg parameter allows for one-pass processing when the signature is being verified. The value of the micalg parameter is dependent on the message digest algorithm(s) used in the calculation of the Message Integrity Check. If multiple message digest algorithms are used, they MUST be separated by commas per [RFC1847]. The values to be placed in the micalg parameter SHOULD be from the following:

Algorithm	Value Used
MD5*	md5
SHA-1*	sha-1
SHA-224	sha-224
SHA-256	sha-256
SHA-384	sha-384
SHA-512	sha-512
Any other	(defined separately in algorithm profile or "unknown" if not defined)

*Note: MD5 and SHA-1 are historical and no longer considered secure. See Appendix B for details.

(Historical note: some early implementations of S/MIME emitted and expected "rsa-md5", "rsa-sha1", and "sha1" for the micalg parameter.) Receiving agents SHOULD be able to recover gracefully from a micalg parameter value that they do not recognize. Future names for this parameter will be consistent with the IANA "Hash Function Textual Names" registry.

3.5.3.3. Sample multipart/signed Message

```
Content-Type: multipart/signed;
    micalg=sha-256;
    boundary="----=_NextBoundry____Fri,_06_Sep_2002_00:25:21";
    protocol="application/pkcs7-signature"

This is a multi-part message in MIME format.

------=_NextBoundry____Fri,_06_Sep_2002_00:25:21

This is some sample content.
------=_NextBoundry____Fri,_06_Sep_2002_00:25:21
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s

MIIBJgYJKoZIhvcNAQcCoIIBFzCCARMCAQExADALBgkqhkiG9w0BBwExgf4w
gfsCAQIwJjASMRAwDgYDVQQDEwdDYXJsUlNBAhBGNGvHgABWvBHTbi7EELOw
MAsGCWCGSAFlAwQCAaAxMC8GCSqGSIb3DQEJBDEiBCCxwpZGNZzTSsugsn+f
lEidzQK4mf/ozKqfmbxhcIkKqjALBgkqhkiG9w0BAQsEgYB0XJV7fjPa5Nuh
oth5msDfP8A5urYUMjhNpWgXG8ae3XpppqVrPi2nVO41onHnkByjkeD/wc31
A9WH8MzFQgSTsrJ65JvffTTXkOpRPxsSHn3wJFwP/atWHkh8YK/jR9bULhUl
Mv5jQEDiwVX5DRasxu6Ld8zv9u5/TsdBNiufGw==

------=_NextBoundry____Fri,_06_Sep_2002_00:25:21--
```

The content that is digested (the first part of the multipart/signed) consists of the bytes:

```
54 68 69 73 20 69 73 20 73 6f 6d 65 20 73 61 6d 70 6c 65 20 63 6f 6e
74 65 6e 74 2e 0d 0a
```

3.6. Creating a Compressed-Only Message

This section describes the format for compressing a MIME entity. Please note that versions of S/MIME prior to version 3.1 did not specify any use of CompressedData, and will not recognize it. The

use of a capability to indicate the ability to receive CompressedData is described in [RFC3274] and is the preferred method for compatibility.

Step 1. The MIME entity to be compressed is prepared according to Section 3.1.

Step 2. The MIME entity and other required data are processed into a CMS object of type CompressedData.

Step 3. The CompressedData object is wrapped in a CMS ContentInfo object.

Step 4. The ContentInfo object is inserted into an application/pkcs7-mime MIME entity.

The smime-type parameter for compressed-only messages is "compressed-data". The file extension for this type of message is ".p7z".

A sample message would be:

```
Content-Type: application/pkcs7-mime; smime-type=compressed-data;
   name=smime.p7z
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7z

eNoLycgsVgCi4vzcVIXixNyCnFSF5Py8ktS8Ej0AlCkKVA==
```

3.7. Multiple Operations

The signed-only, enveloped-only, and compressed-only MIME formats can be nested. This works because these formats are all MIME entities that encapsulate other MIME entities.

An S/MIME implementation MUST be able to receive and process arbitrarily nested S/MIME within reasonable resource limits of the recipient computer.

It is possible to apply any of the signing, encrypting, and compressing operations in any order. It is up to the implementer and the user to choose. When signing first, the signatories are then securely obscured by the enveloping. When enveloping first the signatories are exposed, but it is possible to verify signatures without removing the enveloping. This can be useful in an environment where automatic signature verification is desired, as no private key material is required to verify a signature.

There are security ramifications to choosing whether to sign first or encrypt first. A recipient of a message that is encrypted and then signed can validate that the encrypted block was unaltered, but cannot determine any relationship between the signer and the unencrypted contents of the message. A recipient of a message that is signed then encrypted can assume that the signed message itself has not been altered, but that a careful attacker could have changed the unauthenticated portions of the encrypted message.

When using compression, keep the following guidelines in mind:

- Compression of binary encoded encrypted data is discouraged, since it will not yield significant compression. Base64 encrypted data could very well benefit, however.

- If a lossy compression algorithm is used with signing, you will need to compress first, then sign.

3.8. Creating a Certificate Management Message

The certificate management message or MIME entity is used to transport certificates and/or Certificate Revocation Lists, such as in response to a registration request.

Step 1. The certificates and/or Certificate Revocation Lists are made available to the CMS generating process that creates a CMS object of type SignedData. The SignedData encapContentInfo eContent field MUST be absent and signerInfos field MUST be empty.

Step 2. The SignedData object is wrapped in a CMS ContentInfo object.

Step 3. The ContentInfo object is enclosed in an application/pkcs7-mime MIME entity.

The smime-type parameter for a certificate management message is "certs-only". The file extension for this type of message is ".p7c".

3.9. Registration Requests

A sending agent that signs messages MUST have a certificate for the signature so that a receiving agent can verify the signature. There are many ways of getting certificates, such as through an exchange with a certification authority, through a hardware token or diskette, and so on.

S/MIME v2 [SMIMEv2] specified a method for "registering" public keys with certificate authorities using an application/pkcs10 body part. Since that time, the IETF PKIX Working Group has developed other methods for requesting certificates. However, S/MIME v4.0 does not require a particular certificate request mechanism.

3.10. Identifying an S/MIME Message

Because S/MIME takes into account interoperation in non-MIME environments, several different mechanisms are employed to carry the type information, and it becomes a bit difficult to identify S/MIME messages. The following table lists criteria for determining whether or not a message is an S/MIME message. A message is considered an S/MIME message if it matches any of the criteria listed below.

The file suffix in the table below comes from the "name" parameter in the Content-Type header field, or the "filename" parameter on the Content-Disposition header field. These parameters that give the file suffix are not listed below as part of the parameter section.

Media type	parameters	file suffix
application/pkcs7-mime	n/a	n/a
multipart/signed	protocol= "application/pkcs7-signature"	n/a
application/octet-stream	n/a	p7m, p7s, p7c, p7z

4. Certificate Processing

A receiving agent MUST provide some certificate retrieval mechanism in order to gain access to certificates for recipients of digital envelopes. This specification does not cover how S/MIME agents handle certificates, only what they do after a certificate has been validated or rejected. S/MIME certificate issues are covered in [RFC5750].

At a minimum, for initial S/MIME deployment, a user agent could automatically generate a message to an intended recipient requesting that recipient’s certificate in a signed return message. Receiving and sending agents SHOULD also provide a mechanism to allow a user to "store and protect" certificates for correspondents in such a way so as to guarantee their later retrieval.

4.1. Key Pair Generation

All generated key pairs MUST be generated from a good source of non-deterministic random input [RFC4086] and the private key MUST be protected in a secure fashion.

An S/MIME user agent MUST NOT generate asymmetric keys less than 2048 bits for use with an RSA signature algorithm.

For 2048-bit through 4096-bit RSA with SHA-256 see [RFC5754] and [FIPS186-4]. The first reference provides the signature algorithm's object identifier, and the second provides the signature algorithm's definition.

For RSASSA-PSS with SHA-256, see [RFC4056]. For RSAES-OAEP, see [RFC3560].

4.2. Signature Generation

The following are the requirements for an S/MIME agent generated RSA and RSASSA-PSS signatures:

```
        key size <= 2047 : SHOULD NOT (Note 1)
2048 <= key size <= 4096 : SHOULD     (see Security Considerations)
4096 <  key size         : MAY        (see Security Considerations)
```

Note 1: see Historical Mail Considerations in Section 6.
Note 2: see Security Considerations in Appendix B.

Key sizes for ECDSA and EdDSA are fixed by the curve.

4.3. Signature Verification

The following are the requirements for S/MIME receiving agents during signature verification of RSA and RSASSA-PSS signatures:

```
        key size <= 2047 : SHOULD NOT (Note 1)
2048 <= key size <= 4096 : MUST       (Note 2)
4096 <  key size         : MAY        (Note 2)
```

Note 1: see Historical Mail Considerations in Section 6.
Note 2: see Security Considerations in Appendix B.

Key sizes for ECDSA and EdDSA are fixed by the curve.

4.4. Encryption

The following are the requirements for an S/MIME agent when establishing keys for content encryption using the RSA, and RSA-OAEP algorithms:

```
        key size <= 2047 : SHOULD NOT (Note 1)
2048 <= key size <= 4096 : SHOULD     (Note 2)
4096 <  key size         : MAY        (Note 2)
```

Note 1: see Historical Mail Considerations in Section 6.
Note 2: see Security Considerations in Appendix B.

Key sizes for ECDH are fixed by the curve.

4.5. Decryption

The following are the requirements for an S/MIME agent when establishing keys for content decryption using the RSA and RSAES-OAEP algorithms:

```
        key size <= 2047 : MAY        (Note 1)
2048 <= key size <= 4096 : MUST       (Note 2)
4096 <  key size         : MAY        (Note 2)
```

Note 1: see Historical Mail Considerations in Section 6.
Note 2: see Security Considerations in Appendix B.

Key sizes for ECDH are fixed by the curve.

5. IANA Considerations

The following information updates the media type registration for application/pkcs7-mime and application/pkcs7-signature to refer to this document as opposed to RFC 2311.

Note that other documents can define additional MIME media types for S/MIME.

5.1. Media Type for application/pkcs7-mime

Type name: application

Subtype Name: pkcs7-mime

Required Parameters: NONE

Optional Parameters: smime-type/signed-data
smime-type/enveloped-data
smime-type/compressed-data
smime-type/certs-only
name

Encoding Considerations: See Section 3 of this document

Security Considerations: See Section 6 of this document

Interoperability Considerations: See Sections 1-6 of this document

Published Specification: RFC 2311, RFC 2633, and this document

Applications that use this media type: Security applications

Additional information: NONE

Person & email to contact for further information: iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: NONE

Author: Sean Turner

Change Controller: S/MIME working group delegated from the IESG

5.2. Media Type for application/pkcs7-signature

Type name: application

Subtype Name: pkcs7-signature

Required Parameters: NONE

Optional Parameters: NONE

Encoding Considerations: See Section 3 of this document

Security Considerations: See Section 6 of this document

Interoperability Considerations: See Sections 1-6 of this document

Published Specification: RFC 2311, RFC 2633, and this document

Applications that use this media type: Security applications

Additional information: NONE

Person & email to contact for further information: iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: NONE

Author: Sean Turner

Change Controller: S/MIME working group delegated from the IESG

5.3. Register authEnveloped-data smime-type

IANA is required to register the following value in the "Parameter Values for the smime-type Parameter" registry. The values to be registered are:

smime-type value: authEnveloped-data

Reference: [[This Document, Section 3.2.2]]

6. Security Considerations

Cryptographic algorithms will be broken or weakened over time. Implementers and users need to check that the cryptographic algorithms listed in this document continue to provide the expected level of security. The IETF from time to time may issue documents dealing with the current state of the art. For example:

- The Million Message Attack described in RFC 3218 [RFC3218].

- The Diffie-Hellman "small-subgroup" attacks described in RFC 2785 [RFC2785].

- The attacks against hash algorithms described in RFC 4270 [RFC4270].

This specification uses Public-Key Cryptography technologies. It is assumed that the private key is protected to ensure that it is not accessed or altered by unauthorized parties.

It is impossible for most people or software to estimate the value of a message's content. Further, it is impossible for most people or software to estimate the actual cost of recovering an encrypted message content that is encrypted with a key of a particular size. Further, it is quite difficult to determine the cost of a failed decryption if a recipient cannot process a message's content. Thus, choosing between different key sizes (or choosing whether to just use plaintext) is also impossible for most people or software. However, decisions based on these criteria are made all the time, and therefore this specification gives a framework for using those estimates in choosing algorithms.

The choice of 2048 bits as an RSA asymmetric key size in this specification is based on the desire to provide at least 100 bits of security. The key sizes that must be supported to conform to this specification seem appropriate for the Internet based on [RFC3766]. Of course, there are environments, such as financial and medical systems, that may select different key sizes. For this reason, an implementation MAY support key sizes beyond those recommended in this specification.

Receiving agents that validate signatures and sending agents that encrypt messages need to be cautious of cryptographic processing usage when validating signatures and encrypting messages using keys larger than those mandated in this specification. An attacker could send certificates with keys that would result in excessive cryptographic processing, for example, keys larger than those mandated in this specification, which could swamp the processing element. Agents that use such keys without first validating the certificate to a trust anchor are advised to have some sort of cryptographic resource management system to prevent such attacks.

Some cryptographic algorithms such as RC2 offer little actual security over sending plaintext. Other algorithms such as TripleDES, provide security but are no longer considered to be state of the art. S/MIME requires the use of current state of the art algorithms such

as AES and provides the ability to announce cryptographic capabilities to parties with whom you communicate. This allows the sender to create messages which can use the strongest common encryption algorithm. Using algorithms such as RC2 is never recommended unless the only alternative is no cryptography.

RSA and DSA keys of less than 2048 bits are now considered by many experts to be cryptographically insecure (due to advances in computing power), and should no longer be used to protect messages. Such keys were previously considered secure, so processing previously received signed and encrypted mail will often result in the use of weak keys. Implementations that wish to support previous versions of S/MIME or process old messages need to consider the security risks that result from smaller key sizes (e.g., spoofed messages) versus the costs of denial of service. If an implementation supports verification of digital signatures generated with RSA and DSA keys of less than 1024 bits, it MUST warn the user. Implementers should consider providing different warnings for newly received messages and previously stored messages. Server implementations (e.g., secure mail list servers) where user warnings are not appropriate SHOULD reject messages with weak signatures.

Implementers SHOULD be aware that multiple active key pairs can be associated with a single individual. For example, one key pair can be used to support confidentiality, while a different key pair can be used for digital signatures.

If a sending agent is sending the same message using different strengths of cryptography, an attacker watching the communications channel might be able to determine the contents of the strongly encrypted message by decrypting the weakly encrypted version. In other words, a sender SHOULD NOT send a copy of a message using weaker cryptography than they would use for the original of the message.

Modification of the ciphertext in EnvelopedData can go undetected if authentication is not also used, which is the case when sending EnvelopedData without wrapping it in SignedData or enclosing SignedData within it. This is one of the reasons for moving from EnvelopedData to AuthEnvelopedData, as the authenticated encryption algorithms provide the authentication without needing the SignedData layer.

If an implementation is concerned about compliance with National Institute of Standards and Technology (NIST) key size recommendations, then see [SP800-57].

If messaging environments make use of the fact that a message is signed to change the behavior of message processing (examples would be running rules or UI display hints), without first verifying that the message is actually signed and knowing the state of the signature, this can lead to incorrect handling of the message. Visual indicators on messages may need to have the signature validation code checked periodically if the indicator is supposed to give information on the current status of a message.

Many people assume that the use of an authenticated encryption algorithm is all that is needed for the sender of the message to be authenticated. In almost all cases this is not a correct statement. There are a number of preconditions that need to hold for an authenticated encryption algorithm to provide this service:

- The starting key must be bound to a single entity. The use of a group key only would allow for the statement that a message was sent by one of the entities that held the key but will not identify a specific entity.

- The message must have exactly one sender and one recipient. Having more than one recipient would allow for the second recipient to create a message that the first recipient would believe is from the sender by stripping the second recipient from the message.

- A direct path needs to exist from the starting key to the key used as the content encryption key (CEK). That path needs to guarantees that no third party could have seen the resulting CEK. This means that one needs to be using an algorithm that is called a "Direct Encryption" or a "Direct Key Agreement" algorithm in other contexts. This means that the starting key is used directly as the CEK key, or that the starting key is used to create a secret which then is transformed into the CEK via a KDF step.

S/MIME implementations almost universally use ephemeral-static rather than static-static key agreement and do not use a shared secret for encryption. This means that the first precondition is not met. There is a document [RFC6278] which defined how to use static-static key agreement with CMS, so the first precondition can be met. Currently, all S/MIME key agreement methods derive a KEK and wrap a CEK. This violates the third precondition above. New key agreement algorithms that directly created the CEK without creating an intervening KEK would need to be defined.

Even when all of the preconditions are met and origination of a message is established by the use of an authenticated encryption algorithm, users need to be aware that there is no way to prove this

to a third party. This is because either of the parties can successfully create the message (or just alter the content) based on the fact that the CEK is going to be known to both parties. Thus the origination is always built on a presumption that "I did not send this message to myself."

All of the authenticated encryption algorithms in this document use counter mode for the encryption portion of the algorithm. This means that the length of the plain text will always be known as the cipher text length and the plain text length are always the same. This information can enable passive observers to infer information based solely on the length of the message. Applications for which this is a concern need to provide some type of padding so that the length of the message does not provide this information.

When compression is used with encryption, it has the potential to add an additional layer of security. However, care needs to be taken when designing a protocol that relies on this not to create a compression oracle. Compression oracle attacks require an adaptive input to the process and attack the unknown content of a message based on the length of the compressed output. This means that no attack on the encryption key is necessarily required.

A recent paper on S/MIME and OpenPGP Email security [Efail] has pointed out a number of problems with the current S/MIME specifications and how people have implemented mail clients. Due to the nature of how CBC mode operates, the modes allow for malleability of plaintexts. This malleability allows for attackers to make changes in the cipher text and, if parts of the plain text are known, create arbitrary plaintexts blocks. These changes can be made without the weak integrity check in CBC mode being triggered. This type of attack can be prevented by the use of an AEAD algorithm with a more robust integrity check on the decryption process. It is therefore recommended that mail systems migrate to using AES-GCM as quickly as possible and that the decrypted content not be acted on prior to finishing the integrity check.

The other attack that is highlighted in [Efail] is due to an error in how mail clients deal with HTML and multipart/mixed messages. Clients MUST require that a text/html content type is a complete HTML document (per [RFC1866]). Clients SHOULD treat each of the different pieces of the multipart/mixed construct as being of different origins. Clients MUST treat each encrypted or signed piece of a MIME message as being of different origins both from unprotected content and from each other.

7. References

7.1. Normative References

[ASN.1] "Information Technology - Abstract Syntax Notation (ASN.1)".

ASN.1 syntax consists of the following references [X.680], [X.681], [X.682], and [X.683].

[CHARSETS] "Character sets assigned by IANA.", <http://www.iana.org/assignments/character-sets.>.

[CMS] "Cryptographic Message Syntax".

This is the set of documents dealing with the cryptographic message syntax and refers to [RFC5652] and [RFC5083].

[ESS] "Enhanced Security Services for S/MIME".

This is the set of documents dealing with enhanced security services and refers to [RFC2634] and [RFC5035].

[FIPS186-4] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", Federal Information Processing Standards Publication 186-4, July 2013.

[I-D.ietf-curdle-cms-ecdh-new-curves] Housley, R., "Use of the Elliptic Curve Diffie-Hellman Key Agreement Algorithm with X25519 and X448 in the Cryptographic Message Syntax (CMS)", draft-ietf-curdle-cms-ecdh-new-curves-10 (work in progress), August 2017.

[I-D.ietf-curdle-cms-eddsa-signatures] Housley, R., "Use of EdDSA Signatures in the Cryptographic Message Syntax (CMS)", draft-ietf-curdle-cms-eddsa-signatures-08 (work in progress), October 2017.

[I-D.ietf-lamps-rfc5750-bis] Schaad, J., Ramsdell, B., and S. Turner, "Secure/ Multipurpose Internet Mail Extensions (S/ MIME) Version 4.0 Certificate Handling", draft-ietf-lamps-rfc5750-bis-07 (work in progress), June 2018.

[MIME-SPEC]
"MIME Message Specifications".

This is the set of documents that define how to use MIME. This set of documents is [RFC2045], [RFC2046], [RFC2047], [RFC2049], [RFC6838], and [RFC4289].

[RFC1847] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, DOI 10.17487/RFC1847, October 1995, <https://www.rfc-editor.org/info/rfc1847>.

[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, <https://www.rfc-editor.org/info/rfc2045>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, DOI 10.17487/RFC2047, November 1996, <https://www.rfc-editor.org/info/rfc2047>.

[RFC2049] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, DOI 10.17487/RFC2049, November 1996, <https://www.rfc-editor.org/info/rfc2049>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2183] Troost, R., Dorner, S., and K. Moore, Ed., "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, DOI 10.17487/RFC2183, August 1997, <https://www.rfc-editor.org/info/rfc2183>.

[RFC2634] Hoffman, P., Ed., "Enhanced Security Services for S/MIME", RFC 2634, DOI 10.17487/RFC2634, June 1999, <https://www.rfc-editor.org/info/rfc2634>.

[RFC3274] Gutmann, P., "Compressed Data Content Type for Cryptographic Message Syntax (CMS)", RFC 3274, DOI 10.17487/RFC3274, June 2002, <https://www.rfc-editor.org/info/rfc3274>.

[RFC3370] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, DOI 10.17487/RFC3370, August 2002, <https://www.rfc-editor.org/info/rfc3370>.

[RFC3560] Housley, R., "Use of the RSAES-OAEP Key Transport Algorithm in Cryptographic Message Syntax (CMS)", RFC 3560, DOI 10.17487/RFC3560, July 2003, <https://www.rfc-editor.org/info/rfc3560>.

[RFC3565] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, DOI 10.17487/RFC3565, July 2003, <https://www.rfc-editor.org/info/rfc3565>.

[RFC4056] Schaad, J., "Use of the RSASSA-PSS Signature Algorithm in Cryptographic Message Syntax (CMS)", RFC 4056, DOI 10.17487/RFC4056, June 2005, <https://www.rfc-editor.org/info/rfc4056>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC4289] Freed, N. and J. Klensin, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", BCP 13, RFC 4289, DOI 10.17487/RFC4289, December 2005, <https://www.rfc-editor.org/info/rfc4289>.

[RFC5035] Schaad, J., "Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility", RFC 5035, DOI 10.17487/RFC5035, August 2007, <https://www.rfc-editor.org/info/rfc5035>.

[RFC5083] Housley, R., "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type", RFC 5083, DOI 10.17487/RFC5083, November 2007, <https://www.rfc-editor.org/info/rfc5083>.

[RFC5084] Housley, R., "Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)", RFC 5084, DOI 10.17487/RFC5084, November 2007, <https://www.rfc-editor.org/info/rfc5084>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5753] Turner, S. and D. Brown, "Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)", RFC 5753, DOI 10.17487/RFC5753, January 2010, <https://www.rfc-editor.org/info/rfc5753>.

[RFC5754] Turner, S., "Using SHA2 Algorithms with Cryptographic Message Syntax", RFC 5754, DOI 10.17487/RFC5754, January 2010, <https://www.rfc-editor.org/info/rfc5754>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[SMIMEv4.0] "S/MIME version 4.0".

This group of documents represents S/MIME version 4.0. This set of documents are [RFC2634], [I-D.ietf-lamps-rfc5750-bis], [[This Document]], [RFC5652], and [RFC5035].

[X.680] "Information Technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation. ITU-T Recommendation X.680 (2002)", ITU-T X.680, ISO/IEC 8824-1:2008, November 2008.

[X.681] "Information Technology - Abstract Syntax Notation One (ASN.1): Information object specification", ITU-T X.681, ISO/IEC 8824-2:2008, November 2008.

[X.682] "Information Technology - Abstract Syntax Notation One (ASN.1): Constraint specification", ITU-T X.682, ISO/IEC 8824-3:2008, November 2008.

[X.683] "Information Technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications", ITU-T X.683, ISO/IEC 8824-4:2008, November 2008.

[X.690] "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).", ITU-T X.690, ISO/IEC 8825-1:2002, July 2002.

7.2. Informative References

[Efail] Poddebniak, D., Muller, J., Dresen, C., Ising, F., Schinzel, S., Friedberger, S., Somorovsky, J., and J. Schwenk, "Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels", Work in Progress , May 2018.

[FIPS186-2] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS) [With Change Notice 1]", Federal Information Processing Standards Publication 186-2, January 2000.

[RFC1866] Berners-Lee, T. and D. Connolly, "Hypertext Markup Language - 2.0", RFC 1866, DOI 10.17487/RFC1866, November 1995, <https://www.rfc-editor.org/info/rfc1866>.

[RFC2268] Rivest, R., "A Description of the RC2(r) Encryption Algorithm", RFC 2268, DOI 10.17487/RFC2268, March 1998, <https://www.rfc-editor.org/info/rfc2268>.

[RFC2311] Dusse, S., Hoffman, P., Ramsdell, B., Lundblade, L., and L. Repka, "S/MIME Version 2 Message Specification", RFC 2311, DOI 10.17487/RFC2311, March 1998, <https://www.rfc-editor.org/info/rfc2311>.

[RFC2312] Dusse, S., Hoffman, P., Ramsdell, B., and J. Weinstein, "S/MIME Version 2 Certificate Handling", RFC 2312, DOI 10.17487/RFC2312, March 1998, <https://www.rfc-editor.org/info/rfc2312>.

[RFC2313] Kaliski, B., "PKCS #1: RSA Encryption Version 1.5", RFC 2313, DOI 10.17487/RFC2313, March 1998, <https://www.rfc-editor.org/info/rfc2313>.

[RFC2314] Kaliski, B., "PKCS #10: Certification Request Syntax Version 1.5", RFC 2314, DOI 10.17487/RFC2314, March 1998, <https://www.rfc-editor.org/info/rfc2314>.

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <https://www.rfc-editor.org/info/rfc2315>.

[RFC2630] Housley, R., "Cryptographic Message Syntax", RFC 2630, DOI 10.17487/RFC2630, June 1999, <https://www.rfc-editor.org/info/rfc2630>.

[RFC2631] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, DOI 10.17487/RFC2631, June 1999, <https://www.rfc-editor.org/info/rfc2631>.

[RFC2632] Ramsdell, B., Ed., "S/MIME Version 3 Certificate Handling", RFC 2632, DOI 10.17487/RFC2632, June 1999, <https://www.rfc-editor.org/info/rfc2632>.

[RFC2633] Ramsdell, B., Ed., "S/MIME Version 3 Message Specification", RFC 2633, DOI 10.17487/RFC2633, June 1999, <https://www.rfc-editor.org/info/rfc2633>.

[RFC2785] Zuccherato, R., "Methods for Avoiding the "Small-Subgroup" Attacks on the Diffie-Hellman Key Agreement Method for S/MIME", RFC 2785, DOI 10.17487/RFC2785, March 2000, <https://www.rfc-editor.org/info/rfc2785>.

[RFC3218] Rescorla, E., "Preventing the Million Message Attack on Cryptographic Message Syntax", RFC 3218, DOI 10.17487/RFC3218, January 2002, <https://www.rfc-editor.org/info/rfc3218>.

[RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, DOI 10.17487/RFC3766, April 2004, <https://www.rfc-editor.org/info/rfc3766>.

[RFC3850] Ramsdell, B., Ed., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling", RFC 3850, DOI 10.17487/RFC3850, July 2004, <https://www.rfc-editor.org/info/rfc3850>.

[RFC3851] Ramsdell, B., Ed., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, DOI 10.17487/RFC3851, July 2004, <https://www.rfc-editor.org/info/rfc3851>.

[RFC3852] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, DOI 10.17487/RFC3852, July 2004, <https://www.rfc-editor.org/info/rfc3852>.

[RFC4134] Hoffman, P., Ed., "Examples of S/MIME Messages", RFC 4134, DOI 10.17487/RFC4134, July 2005, <https://www.rfc-editor.org/info/rfc4134>.

[RFC4270] Hoffman, P. and B. Schneier, "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, DOI 10.17487/RFC4270, November 2005, <https://www.rfc-editor.org/info/rfc4270>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC5750] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling", RFC 5750, DOI 10.17487/RFC5750, January 2010, <https://www.rfc-editor.org/info/rfc5750>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <https://www.rfc-editor.org/info/rfc5751>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <https://www.rfc-editor.org/info/rfc6151>.

[RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <https://www.rfc-editor.org/info/rfc6194>.

[RFC6278] Herzog, J. and R. Khazan, "Use of Static-Static Elliptic Curve Diffie-Hellman Key Agreement in Cryptographic Message Syntax", RFC 6278, DOI 10.17487/RFC6278, June 2011, <https://www.rfc-editor.org/info/rfc6278>.

[RFC7114] Leiba, B., "Creation of a Registry for smime-type Parameter Values", RFC 7114, DOI 10.17487/RFC7114, January 2014, <https://www.rfc-editor.org/info/rfc7114>.

[RFC7905] Langley, A., Chang, W., Mavrogiannopoulos, N., Strombergson, J., and S. Josefsson, "ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)", RFC 7905, DOI 10.17487/RFC7905, June 2016, <https://www.rfc-editor.org/info/rfc7905>.

[SMIMEv2] "S/MIME version v2".

This group of documents represents S/MIME version 2. This set of documents are [RFC2311], [RFC2312], [RFC2313], [RFC2314], and [RFC2315].

[SMIMEv3] "S/MIME version 3".

This group of documents represents S/MIME version 3. This set of documents are [RFC2630], [RFC2631], [RFC2632], [RFC2633], [RFC2634], and [RFC5035].

[SMIMEv3.1]
"S/MIME version 3.1".

This group of documents represents S/MIME version 3.1. This set of documents are [RFC2634], [RFC3850], [RFC3851], [RFC3852], and [RFC5035].

[SMIMEv3.2]
"S/MIME version 3.2".

This group of documents represents S/MIME version 3.2. This set of documents are [RFC2634], [RFC5750], [RFC5751], [RFC5652], and [RFC5035].

[SP800-56A]
National Institute of Standards and Technology (NIST), "Special Publication 800-56A Revision 2: Recommendation Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", May 2013.

[SP800-57]
National Institute of Standards and Technology (NIST), "Special Publication 800-57: Recommendation for Key Management", August 2005.

[TripleDES]
Tuchman, W., "Hellman Presents No Shortcut Solutions to DES"", IEEE Spectrum v. 16, n. 7, pp 40-41, July 1979.

Appendix A. ASN.1 Module

Note: The ASN.1 module contained herein is unchanged from RFC 3851 [SMIMEv3.1] with the exception of a change to the prefersBinaryInside ASN.1 comment. This module uses the 1988 version of ASN.1.

```
SecureMimeMessageV3dot1

  { iso(1) member-body(2) us(840) rsadsi(113549)
```

```
            pkcs(1) pkcs-9(9) smime(16) modules(0) msg-v3dot1(21) }

   DEFINITIONS IMPLICIT TAGS ::=

   BEGIN

   IMPORTS

   -- Cryptographic Message Syntax [CMS]
      SubjectKeyIdentifier, IssuerAndSerialNumber,
      RecipientKeyIdentifier
          FROM  CryptographicMessageSyntax
                { iso(1) member-body(2) us(840) rsadsi(113549)
                  pkcs(1) pkcs-9(9) smime(16) modules(0) cms-2001(14) };

   --  id-aa is the arc with all new authenticated and unauthenticated
   --  attributes produced by the S/MIME Working Group

   id-aa OBJECT IDENTIFIER ::= {iso(1) member-body(2) usa(840)
           rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) attributes(2)}

   -- S/MIME Capabilities provides a method of broadcasting the
   -- symmetric capabilities understood.  Algorithms SHOULD be ordered
   -- by preference and grouped by type

   smimeCapabilities OBJECT IDENTIFIER ::= {iso(1) member-body(2)
           us(840) rsadsi(113549) pkcs(1) pkcs-9(9) 15}

   SMIMECapability ::= SEQUENCE {
      capabilityID OBJECT IDENTIFIER,
      parameters ANY DEFINED BY capabilityID OPTIONAL }

   SMIMECapabilities ::= SEQUENCE OF SMIMECapability

   -- Encryption Key Preference provides a method of broadcasting the
   -- preferred encryption certificate.

   id-aa-encrypKeyPref OBJECT IDENTIFIER ::= {id-aa 11}

   SMIMEEncryptionKeyPreference ::= CHOICE {
      issuerAndSerialNumber   [0] IssuerAndSerialNumber,
      receipentKeyId          [1] RecipientKeyIdentifier,
      subjectAltKeyIdentifier [2] SubjectKeyIdentifier
   }

   -- receipentKeyId is spelt incorrectly, but kept for historical
   -- reasons.
```

```
   id-smime OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
           rsadsi(113549) pkcs(1) pkcs9(9) 16 }

   id-cap  OBJECT IDENTIFIER ::= { id-smime 11 }

   -- The preferBinaryInside OID indicates an ability to receive
   -- messages with binary encoding inside the CMS wrapper.
   -- The preferBinaryInside attribute's value field is ABSENT.

   id-cap-preferBinaryInside  OBJECT IDENTIFIER ::= { id-cap 1 }

   --  The following list OIDs to be used with S/MIME V3

   -- Signature Algorithms Not Found in [RFC3370], [RFC5754], [RFC4056],
   -- and [RFC3560]

   --
   -- md2WithRSAEncryption OBJECT IDENTIFIER ::=
   --    {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1)
   --     2}

   --
   -- Other Signed Attributes
   --
   -- signingTime OBJECT IDENTIFIER ::=
   --    {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
   --     5}
   --    See [CMS] for a description of how to encode the attribute
   --    value.

   SMIMECapabilitiesParametersForRC2CBC ::= INTEGER
   --        (RC2 Key Length (number of bits))

   END
```

Appendix B. Historic Mail Considerations

Over the course of updating the S/MIME specifications, the set of recommended algorithms has been modified each time the document has been updated. This means that if a user has historic emails and their user agent has been updated to only support the current set of recommended algorithms some of those old emails will no longer be accessible. It is strongly suggested that user agents implement some of the following algorithms for dealing with historic emails.

This appendix contains a number of references to documents that have been obsoleted or replaced. This is intentional as frequently the updated documents do not have the same information in them.

B.1. DigestAlgorithmIdentifier

The following algorithms have been called our for some level of support by previous S/MIME specifications:

- SHA-1 was dropped in [SMIMEv4.0]. SHA-1 is no longer considered to be secure as it is no longer collision-resistant. The IETF statement on SHA-1 can be found in [RFC6194] but it is out-of-date relative to the most recent advances.

- MD5 was dropped in [SMIMEv4.0]. MD5 is no longer considered to be secure as it is no longer collision-resistant. Details can be found in [RFC6151].

B.2. Signature Algorithms

There are a number of problems with validating signatures on sufficiently historic messages. For this reason it is strongly suggested that UAs treat these signatures differently from those on current messages. These problems include:

- CAs are not required to keep certificates on a CRL beyond one update after a certificate has expired. This means that unless CRLs are cached as part of the message it is not always possible to check if a certificate has been revoked. The same problems exist with OCSP responses as they may be based on a CRL rather than on the certificate database.

- RSA and DSA keys of less than 2048 bits are now considered by many experts to be cryptographically insecure (due to advances in computing power). Such keys were previously considered secure, so processing of historic signed messages will often result in the use of weak keys. Implementations that wish to support previous versions of S/MIME or process old messages need to consider the security risks that result from smaller key sizes (e.g., spoofed messages) versus the costs of denial of service.

  [SMIMEv3.1] set the lower limit on suggested key sizes for creating and validation at 1024 bits. Prior to that the lower bound on key sizes was 512 bits.

- Hash functions used to validate signatures on historic messages may longer be considered to be secure. (See below.) While there are not currently any known practical pre-image or second pre-image attacks against MD5 or SHA-1, the fact they are no longer considered to be collision resistant implies that the security levels of the signatures are generally considered suspect. If a message is known to be historic, and it has been in the possession

of the client for some time, then it might still be considered to be secure.

- The previous two issues apply to the certificates used to validate the binding of the public key to the identity that signed the message as well.

The following algorithms have been called out for some level of support by previous S/MIME specifications:

- RSA with MD5 was dropped in [SMIMEv4.0]. MD5 is no longer considered to be secure as it is no longer collision-resistant. Details can be found in [RFC6151].

- RSA and DSA with SHA-1 were dropped in [SMIMEv4.0]. SHA-1 is no longer considered to be secure as it is no longer collision-resistant. The IETF statement on SHA-1 can be found in [RFC6194] but it is out-of-date relative to the most recent advances.

- DSA with SHA-256 was dropped in [SMIMEv4.0]. DSA has been replaced by elliptic curve versions.

As requirements for mandatory to implement has changed over time, some issues have been created that can cause interoperability problems:

- S/MIME v2 clients are only required to verify digital signatures using the rsaEncryption algorithm with SHA-1 or MD5, and might not implement id-dsa-with-sha1 or id-dsa at all.

- S/MIME v3 clients might only implement signing or signature verification using id-dsa-with-sha1, and might also use id-dsa as an AlgorithmIdentifier in this field.

- Note that S/MIME v3.1 clients support verifying id-dsa-with-sha1 and rsaEncryption and might not implement sha256withRSAEncryption.

NOTE: Receiving clients SHOULD recognize id-dsa as equivalent to id-dsa-with-sha1.

For 512-bit RSA with SHA-1 see [RFC3370] and [FIPS186-2] without Change Notice 1, for 512-bit RSA with SHA-256 see [RFC5754] and [FIPS186-2] without Change Notice 1, and for 1024-bit through 2048-bit RSA with SHA-256 see [RFC5754] and [FIPS186-2] with Change Notice 1. The first reference provides the signature algorithm’s object identifier, and the second provides the signature algorithm’s definition.

For 512-bit DSA with SHA-1 see [RFC3370] and [FIPS186-2] without Change Notice 1, for 512-bit DSA with SHA-256 see [RFC5754] and [FIPS186-2] without Change Notice 1, for 1024-bit DSA with SHA-1 see [RFC3370] and [FIPS186-2] with Change Notice 1, for 1024-bit and above DSA with SHA-256 see [RFC5754] and [FIPS186-4]. The first reference provides the signature algorithm's object identifier and the second provides the signature algorithm's definition.

B.3. ContentEncryptionAlgorithmIdentifier

The following algorithms have been called out for some level of support by previous S/MIME specifications:

- RC2/40 [RFC2268] was dropped in [SMIMEv3.2]. The algorithm is known to be insecure and, if supported, should only be used to decrypt existing email.

- DES EDE3 CBC [TripleDES], also known as "tripleDES" is dropped in [SMIMEv4.0]. This algorithms is removed from the supported list due to the fact that it has a 64-bit block size and the fact that it offers less that 128-bits of security. This algorithm should be supported only to decrypt existing email, it should not be used to encrypt new emails.

B.4. KeyEncryptionAlgorithmIdentifier

The following algorithms have been called out for some level of support by previous S/MIME specifications:

- DH ephemeral-static mode, as specified in [RFC3370] and [SP800-57], was dropped in [SMIMEv4.0].

- RSA key sizes have been increased over time. Decrypting old mail with smaller key sizes is reasonable, however new mail should use the updated key sizes.

For 1024-bit DH, see [RFC3370]. For 1024-bit and larger DH, see [SP800-56A]; regardless, use the KDF, which is from X9.42, specified in [RFC3370].

Appendix C. Moving S/MIME v2 Message Specification to Historic Status

The S/MIME v3 [SMIMEv3], v3.1 [SMIMEv3.1], and v3.2 [SMIMEv3.2] are backwards compatible with the S/MIME v2 Message Specification [SMIMEv2], with the exception of the algorithms (dropped RC2/40 requirement and added DSA and RSASSA-PSS requirements). Therefore, it is recommended that RFC 2311 [SMIMEv2] be moved to Historic status.

Appendix D. Acknowledgments

Many thanks go out to the other authors of the S/MIME version 2 Message Specification RFC: Steve Dusse, Paul Hoffman, Laurence Lundblade, and Lisa Repka. Without v2, there wouldn't be a v3, v3.1, v3.2 or v4.0.

Some of the examples in this document were copied from [RFC4134]. Thanks go the the people who wrote and verified the examples in that document.

A number of the members of the S/MIME Working Group have also worked very hard and contributed to this document. Any list of people is doomed to omission, and for that I apologize. In alphabetical order, the following people stand out in my mind because they made direct contributions to various versions of this document:

Tony Capel, Piers Chivers, Dave Crocker, Bill Flanigan, Peter Gutmann, Alfred Hoenes, Paul Hoffman, Russ Housley, William Ottaway, and John Pawling.

The version 4 update to the S/MIME documents was done under the auspices of the LAMPS Working Group.

Authors' Addresses

Jim Schaad
August Cellars

Email: ietf@augustcellars.com

Blake Ramsdell
Brute Squad Labs, Inc.

Email: blaker@gmail.com

Sean Turner
sn3rd

Email: sean@sn3rd.com

Network Working Group
Internet Draft
Intended Status: Standards Track
Expires: April 15, 2018

Sean Turner
sn3rd

October 12, 2017

EST (Enrollment over Secure Transport) Extensions
draft-turner-est-extensions-11.txt

Abstract

The EST (Enrollment over Secure Transport) protocol defined a Well-Known URI (Uniform Resource Identifier): /.well-known/est along with a number of other path components that clients use for PKI (Public Key Infrastructure) services, namely certificate enrollment (e.g., /simpleenroll). This document defines a number of other PKI services as additional path components, specifically firmware and trust anchors as well as symmetric, asymmetric, and encrypted keys. This document also specifies the PAL (Package Availability List), which is an XML (Extensible Markup Language) file or JSON (JavaScript Object Notation) object that clients use to retrieve packages available and authorized for them. This document extends the EST server path components to provide these additional services.

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."

Copyright Notice

Copyright (c) 2017 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.

Table of Contents

1. Introduction

The EST (Enrollment over Secure Transport) protocol [RFC7030] defines the Well-Known URI (Uniform Resource Identifier) /.well-known/est to support selected PKI (Public Key Infrastructure) related services with path components (PCs) such as simple enrollment with /simpleenroll, rekey or renew with /simplereenroll, etc. A server that wishes to support additional PKI-related services and other security-related packages could use the same .well-known URI by defining additional PCs. This document defines six such PCs:

- /pal - The PAL (Package Availability List) provides a list of all known packages available and authorized for a client. By accessing the service provided by this PC first, the client can walk through the PAL and download all the packages necessary to begin operating securely. The PAL essentially points to other PCs including the ones defined in this document as well as those defined in [RFC7030], which include /cacerts, /simpleenroll, /simplereenroll, /fullcmc, /serverkeygen, and /csrattrs. The /pal PC is described in Section 2.

- /eecerts - EE (End-Entity) certificates [RFC5280] are needed by the client when they invoke a security protocol for communicating with a peer (i.e., they become operational and do something

meaningful as opposed to just communicating with the infrastructure). If the infrastructure knows the certificate(s) needed by the client, then providing the peer's certificate avoids the client having to discover the peer's certificate. This service is not meant to be a general purpose repository to which clients query a "repository" and then get a response; this is purely a push mechanism. The /eecerts PC is described in Section 3.

- /crls - CRLs (Certificate Revocation Lists) and Authority Revocation Lists (ARLs) [RFC5280] are also needed by the client when they validate certificate paths. CRLs (and ARLs) from TAs (Trust Anchors) and intermediate CAs (Certification Authorities) are needed to validate the certificates used to generate the client's certificate or the peer's certificate, which is provided by the /eecerts PC, and providing them saves the client from having to "discover" them and then retrieve them. CRL "discovery" is greatly aided by the inclusion of the CRL Distribution Point certificate extension [RFC5280], but this extension is not always present in certificates and requires another connection to retrieve them. Like the /eecerts PC, this service is not meant to be a general purpose repository to which clients query a repository and then get a response; this is purely a push mechanism. The /crls PC is described in Section 4.

- /symmetrickeys - In some cases, clients use symmetric keys [RFC6031] when communicating with their peers. If the client's peers are known by the server a priori, then providing them saves the client or an administrator from later having to find, retrieve and install them. Like the /eecerts and /crls PCs, this service is not meant to be a general purpose repository to which clients query a repository and then get a response; this is purely a push mechanism for the keys themselves. However, things do not always go as planned and clients need to inform the server about any errors. If things did go well, then the client, if requested, needs to provide a receipt [RFC7191]. The /symmetrickeys and /symmetrickeys/return PCs are described in Section 5.

- /firmware - Some client firmware and software support automatic update mechanisms and some do not. For those that do not, the /firmware PC provides a mechanism for the infrastructure to inform the client that firmware and software updates [RFC4108] are available. Because updates do not always go as planned and because sometimes the server needs to know whether the firmware was received and processed, this PC also provides a mechanism to return errors and receipts. The /firmware and /firmware/return PCs are defined in Section 6.

- /tamp - To control the TAs in client TA databases, servers use the /tamp PC to request that clients retrieve a TAMP (Trust Anchor Management Protocol) query, update, and adjust packages [RFC5934] and clients use the /tamp/return PC to return TAMP response, confirm, and error [RFC5934]. The /tamp and /tamp/return PCs are defined in Section 7.

This document also extends the /est/serverkeygen PC [RFC7030] to support (see Section 8):

- Returning asymmetric key package receipts and errors [RFC7191].
- Encapsulating returned asymmetric keys in additional CMS content types [RFC7193].
- Returning server-generated public key pairs encapsulated in PKCS#12 [RFC7292].

While the motivation is to provide packages to clients during enrollment so that they can perform securely after enrollment, the services defined in this specification can be used after enrollment.

1.1. Definitions

Familiarity with Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages [RFC4108], Certificate Management over CMS (CMC) [RFC5272], Cryptographic Message Syntax (CMS) Encrypted Key Package [RFC6032], Cryptographic Message Syntax (CMS) [RFC5652][RFC6268], Trust Anchor Management Protocol (TAMP) [RFC5934], Cryptographic Message Syntax (CMS) Content Constraints Extension [RFC6010], CMS Symmetric Key Package Content Type [RFC6031], Enrollment over Secure Transport protocol [RFC7030], CMS Key Package Receipt and Error Content Types [RFC7191] is assumed. Also, familiarity with the CMS protecting content types signed data and encrypted data is assumed; CMS signed data and encrypted data are defined in [RFC5652] and CMS encrypted key package is defined in [RFC6032].

In addition to the definitions found in [RFC7030], the following definitions are used in this document:

Agent: An entity that performs functions on behalf of a client. Agents can service a) one or more clients on the same network as the server, b) clients on non-IP based networks, or c) clients that have a non-electronic air gap [RFC4949] between themselves and the server. Interactions between the agent and client in the last two cases are beyond the scope of this document. Before an agent can service clients, the agent must have a trust relationship with the server, be authorized to act on behalf of clients.

Client: A device that ultimately consumes and uses the packages to enable communications. In other words, the client is the end-point for the packages and an agent may have one or more clients. To avoid confusion, this document henceforth uses the term client to refer to both agents and clients.

Package: An object that contains one or more content types. There are numerous types of packages: Asymmetric Keys, Symmetric Keys, Encrypted Keys, CRLs, Public Key Certificate Management, Firmware, Public Key Certificates, and TAMP packages. All of these packages are digitally signed by their creator and encapsulated in a CMS signed data [RFC5652][RFC6268] (except the public key certificates and CRLs that are already digitally signed by a CA); Firmware receipts and errors, TAMP responses, confirms, and errors, as well as Key Package receipts and errors that can be optionally signed. Certificate and CRLs are included in a package that uses signed data, which is often referred to as a degenerate CMS or "certs-only" or "crls-only" message [RFC5751][RFC6268], but no signature or content is present; hence the name certs-only and crls-only.

Note: As per [RFC7030], the creator may or may not be the EST server or the EST CA.

1.2. Authentication and Authorization

Client and server authentication as well as client and server authorization are as defined in [RFC7030]. The requirements for each are discussed in the request and response sections of each of the PCs defined by this document.

The requirements for the TA databases are as specified in [RFC7030] as well.

1.3. TLS Cipher Suites

TLS cipher suite and issues associated with them are as defined in [RFC7030].

1.4. URI Configuration

As specified in Section 3.1 of [RFC7030], the client is configured with sufficient information to form the server URI [RFC3986]. Like EST, this configuration mechanism is beyond the scope of this document.

1.5. Message Types

This document uses existing media types for the messages as specified

by "Internet X.509 Public Key Infrastructure Protocol: FTP and HTTP" [RFC2585], "The application/pkcs10 Media Type" [RFC5967], and CMC [RFC5272].

For consistency with [RFC5273], each distinct EST message type uses an HTTP Content-Type header with a specific media type.

The EST messages and their corresponding media types for each operation are:

Message type (per operation)	Request media type Response media type(s) Source(s) of types	Request section(s) Response section
Locate Available Packages /pal	N/A application/xml or application/json [RFC7303][RFC7159]	Section 2.2 Section 2.3
Distribute EE Certificates /eecerts	N/A application/pkcs7-mime [RFC5751]	Section 3.1 Section 3.2
Distribute CRLs /crls	N/A application/pkcs7-mime [RFC5751]	Section 4.1 Section 4.2
Symmetric Key Distribution /symmetrickeys	N/A application/cms [RFC7193]	Section 5.1.1 Section 5.1.2
Return Symmetric Key Receipts/Errors /symmetrickeys/return	application/cms N/A [RFC7193]	Section 5.2.1 Section 5.2.2
Firmware Distribution /firmware	N/A application/cms [RFC7193]	Section 6.1.1 Section 6.1.2

Return Firmware Receipts/Errors /firmware/return	application/cms N/A [RFC7193]	Section 6.2.1 Section 6.2.2
Trust Anchor Management /tamp	N/A application/ tamp-status-query tamp-update tamp-apex-update tamp-community-update tamp-sequence-adjust [RFC5934]	Section 7.1.1 Section 7.1.2
Return TAMP Responses/ Confirms/ Errors /tamp/return	application/ tamp-status-query-response tamp-update-confirm tamp-apex-update-confirm tamp-community-update-confirm tamp-sequence-adjust-confirm tamp-error N/A [RFC5934]	Section 7.2.1 Section 7.2.2
Server-Side Key Generation /serverkeygen	application/pkcs10 with content type attribute CSR application/cms [RFC7193]	Section 8.1 Section 8.1
Return Asymmetric Key Receipts/Errors /serverkeygen/ return	application/cms N/A [RFC7193]	Section 8.2 Section 8.2
Server-Side Key Generation: PKCS#12 /serverkeygen	application/pkcs10 application/pkcs12 [RFC7193]	Section 8.3.1 Section 8.3.2

1.6. Key Words

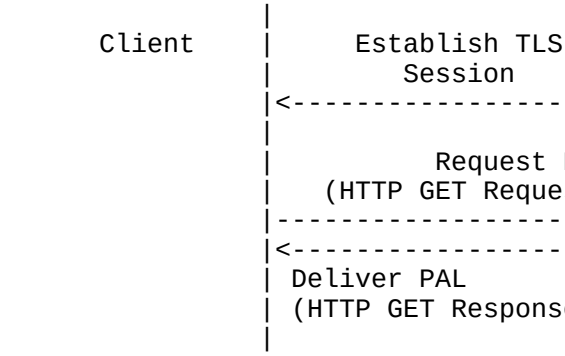
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Locate Available Packages

The PAL (Package Availability List) is either an XML (Extensible Markup Language) [XML] or JSON (JavasScript Object Notation) [RFC7159] object available through the /pal PC that furnishes the following information to clients:

- o Advertisements for available packages that can be retrieved from the server;
- o Notifications to begin public key certificate management or to return package receipts and errors; and
- o Advertisement for another PAL.

After being configured (see Section 1.4), the client can use this service to retrieve their PAL (see Section 2.1) that, if properly constructed (see Section 2.3), allows the client to determine some or all of the security-related packages needed for bootstrapping. Each PAL entry refers to other PCs, defined in this document as well as those defined in [RFC7030], that clients use to retrieve packages, e.g., CA certificates, firmware, trust anchors, symmetric keys, and asymmetric keys, available for it or to be notified to initiate public key certificate enrollment. PAL entries can also be used to notify clients they are to return receipts or errors for certain packages (see Section 2.1.1). Placing these entries after entries that clients used to retrieve the packages is the same as requesting receipts in the originally distributed package. Figure 1 provides a ladder diagram for the /pal PC protocol flow. Appendix A provides a detailed example.



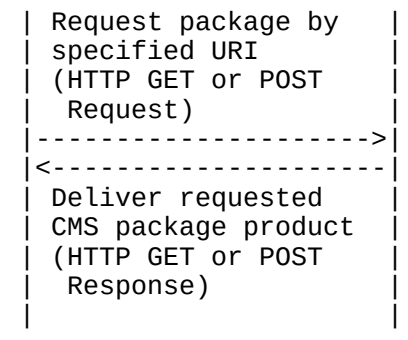


Figure 1 - /pal Message Sequence

PALs are designed to support an arbitrary number of entries, but for PALs that need to be divided for whatever reason there is a special PAL entry type, which are collectively referred to as PAL Package Types (see Sections 2.1 and 2.1.1), number 0001 is defined that refers to another PAL. If present, the 0001 package type is always last because other entries after it are ignored. Also, the 0001 package type cannot be the only PAL entry to avoid needlessly dereferencing URIs.

In addition to using the PAL during bootstrapping, clients can be configured to periodically poll the server to determine if there are updated packages available for it. Note that the mechanism to configure how often clients poll the server is out-of-scope. However, there are some services that support indicating when to return (e.g., simple enrollment and re-enroll responses include the Retry-After header [RFC7030]).

As noted earlier, the PAL support two variants: XML and JSON. Clients include the HTTP Accept header [RFC7231] when they connect to the server to indicate whether they support XML or JSON.

The client MUST authenticate the server as specified in [RFC7030] and the client MUST check the server’s authorization as specified in [RFC7030].

The server MUST authenticate the client as specified in [RFC7030] and the server MUST check the client’s authorization as specified in [RFC7030].

PAL support is OPTIONAL. It is shown in figures throughout this document but clients need not support the PAL to access services offered by the server.

2.1. PAL Format

Each PAL is composed of zero or more entries. Each entry is composed of four fields, type, date, size, and info, whose semantics follow:

Note: Both XML elements and JSON values are described below. XML elements are enclosed in angle brackets <> and JSON values are enclosed in single quotes ''. When described together they are enclosed in brackets [] separated by |.

- o [<type> | 'type'] uniquely identifies each package that a client may retrieve from the server with a 4-digit string. [<type> | 'type'] MUST be present. The PAL Package Types are defined in Section 2.1.1.

- o [<date> | 'date'] either indicates:

  - * The date and time that the client last successfully downloaded the identified package from the server. [<date> | 'date'] MUST be represented as Generalized Time with 20 characters: YYYY-MM-DDTHH:MM:SSZ; <date> matches the dateTime production in "canonical representation" [XMLSCHEMA]; 'date' is a string. Implementations SHOULD NOT rely on time resolution finer than seconds and MUST NOT generate time instants that specify leap seconds.

  - * The omission of [<date> | 'date'] indicates that:

    - There is no indication the client has successfully downloaded the identified package, or

    - The PAL entry corresponds to a pointer to the next PAL or the server is requesting a package from the client (e.g., certification request, receipt, error).

- o [<size> | 'size'] indicates the size in bytes of the package; <size> is a nonNegativeInteger and 'size' is a number. A package size of zero (i.e., "0" without the quotes) indicates that the client needs to begin a transaction or return an error or receipt. [<size> | 'size'] MUST be present.

- o [<info> | 'info'] provides either an SKI (Subject Key Identifier), a DN (Distinguished Name), an Issuer and Serial Number tuple or a URI, i.e., it is a choice between these four all of which are defined in [RFC5280]. When a URI [RFC3986] is included, [<uri> | 'uri'] indicates the location where the identified package can be retrieved. When a DN, an SKI, or an Issuer Name and Serial Number tuple is included it points to a

certificate that is the subject of the notification (i.e., the certificate to be rekeyed or renewed); [<dn> | 'dn'] is encoded as a string with the format defined in [RFC4514]; <ski> is a hexBinary and 'ski' is a string of hex digits (i.e., 0-9, a-f, and A-F); [<iasn> | 'iasn'] includes both [<issuer> | 'issuer'] and [<serial> | 'serial'] as a complexType in XML and an object in JSON. [<issuer> | 'issuer'] is a DN encoded as a string with the format defined in [RFC4514]; <serial> is a positiveInteger and 'serial' is a number. [<info> | 'info'] MUST be present and [<info> | 'info'] MUST include exactly one [<dn> | 'dn'], [<ski> | 'ski'], [<iasn> | 'iasn'], or [<uri> | 'uri'].

Clients are often limited by the size of objects they can consume, the PAL is not immune to these limitations. As opposed to picking a limit for all clients, a special package type is defined, see Section 2.1.1, to indicate that another PAL is available. Servers can use this value to limit the size of the PALs provided to clients. The mechanism for servers to know client PAL size limits is beyond the scope of the document; one possible solution is through provisioned information.

2.1.1. PAL Package Types

Table 1 lists the PAL package types that are defined by this document:

NOTE: CSR is Certificate Signing Request, DS is Digital Signature and KE is Key Establishment.

Package Number	Package Description
0000:	Reserved
0001:	Additional PAL value present
0002:	X.509 CA certificate
0003:	X.509 EE certificate
0004:	X.509 ARL
0005:	X.509 CRL
0006:	Start DS certificate enrollment with CSR attribute
0007:	Start DS certificate enrollment
0008:	DS certificate enrollment (success)
0009:	DS certificate enrollment (failure)
0010:	Start DS certificate re-enrollment
0011:	DS certificate re-enrollment (success)
0012:	DS certificate re-enrollment (failure)
0013:	Start KE certificate enrollment with CSR attribute
0014:	Start KE certificate enrollment
0015:	KE certificate enrollment (success)

0016:	KE certificate enrollment (failure)
0017:	Start KE certificate re-enrollment
0018:	KE certificate re-enrollment (success)
0019:	KE certificate re-enrollment (failure)
0020:	Asymmetric Key Package (PKCS#8)
0021:	Asymmetric Key Package (CMS)
0022:	Asymmetric Key Package (PKCS#12)
0023:	Asymmetric Key Package Receipt or Error
0024:	Symmetric Key Package
0025:	Symmetric Key Package Receipt or Error
0026:	Firmware Package
0027:	Firmware Package Receipt or Error
0028:	TAMP Status Query
0029:	TAMP Status Query Response or Error
0030:	Trust Anchor Update
0031:	Trust Anchor Update Confirm or Error
0032:	Apex Trust Anchor Update
0033:	Apex Trust Anchor Update Confirm or Error
0034:	Community Update
0035:	Community Update Confirm or Error
0036:	Sequence Number Adjust
0037:	Sequence Number Adjust Confirm or Error

Table 1 - PAL Package Types

PAL package types are essentially hints about the type of package the client is about to retrieve or is asked to return. Savvy clients can parse the packages to determine what has been provided, but in some instances it is better to know before retrieving the package. The hint provided here does not obviate the need for clients to check the type of package provided before they store it possibly in specially allocated locations (i.e., some clients might store Root ARLs separately from intermediate CRLs). For packages provided by the client, the server is asking the client to provide an enrollment package, receipt, response, confirm or error.

The PAL package types have the following meaning:

NOTE: The semantics behind Codes 0002 and 0006-0021 are defined in [RFC7030].

0000 Reserved: Reserved for future use.

0001 Additional PAL value present: Indicates that this PAL entry refers to another PAL by referring to another /pal URI, which is defined in this section. This PAL package type limits the size of PALs to a more manageable size for clients. If this PAL Package Type appears it MUST be the last entry in the PAL.

Additionally, this PAL Package Type MUST NOT the only entry to avoid endless dereferencing URIs.

0002 X.509 CA certificate: Indicates that one or more CA certificates [RFC5280] are available for the client by pointing to a /cacerts URI, which is defined in [RFC7030].

0003 X.509 EE certificate: Indicates that one or more EE certificate [RFC5280] is available for the client by pointing to an /eecerts URI, which is defined in Section 3.

0004 X.509 ARL: Indicates that one or more ARL (Authority Revocation List) [RFC5280] is available for the client by pointing to a /crls URI, which is defined in Section 4.

0005 X.509 CRL: Indicates that one or more CRL (Certificate Revocation List) [RFC5280] is available for the client by pointing to a /crls URI, which is defined in Section 4.

NOTE: See Section 9 for additional information about PAL and certificate enrollment interaction. See Appendix B for additional informative information.

0006 Start DS (Digital Signature) certificate enrollment with CSR: Indicates that the client begin enrolling their DS certificate (i.e., those certificates for which the key usage extension will have digital signature set) using a template provided by the server with a CSR (Certificate Signing Request) attribute (see Appendix B). The PAL entry points to a /csrattrs URI, which is defined in [RFC7030].

0007 Start DS (Digital Signature) certificate enrollment: Indicates that the client begin enrolling their DS certificate. The PAL entry points to a /simpleenroll URI, which is defined in [RFC7030].

0008 DS certificate enrollment (success): Indicates that the client retrieve a successful certification response. The PAL entry points to a /simpleenroll or a /fullcmc URI, which are both defined in [RFC7030].

0009 DS certificate enrollment (failure): Indicates that the client retrieve a failed certification response for a DS certificate. This PAL entry points to a /simpleenroll or a /fullcmc URI.

0010 Start DS certificate re-enrollment: Indicates that the client rekey or renew a DS certificate. The PAL entry points to a /simplereenroll or a /fullcmc URI.

0011 DS certificate re-enrollment (success): See PAL package type 0008.

0012 DS certificate re-enrollment (failure): See PAL package type 0009.

NOTE: The KE (Key Establishment) responses that follow use the same URIs as DS certificates except in the requested certificates the key usage extension request will have only either key agreement or key transport set.

0013 Start KE certificate enrollment with CSR: See PAL package type 0006.

0014 Start KE certificate enrollment: See PAL package type 0007.

0015 KE certificate enrollment (success): See PAL package type 0008.

0016 KE certificate enrollment (failure): See PAL package type 0009.

0017 Start KE certificate re-enrollment: See PAL package type 0010.

0018 KE certificate re-enrollment (success): See PAL package type 0008.

0019 KE certificate re-enrollment (failure): See PAL package type 0009.

NOTE: The variations on the asymmetric key packages is due to the number of CMS content types that can be used to protect the asymmetric key; the syntax for the asymmetric key is the same but additional ASN.1 is needed to include it in a signed data (i.e., the ASN.1 needs to be a CMS content type not the private key info type). See Section 8 of this document for additional information.

0020 Asymmetric Key Package (PKCS#8): Indicates that an asymmetric key generated by the server is available for the client; the package is an asymmetric key without additional encryption as specified in Section 4.4.2 of [RFC7030]. The PAL entry points to a /serverkeygen or a /fullcmc URI, which are defined in [RFC7030].

0021 Asymmetric Key Package (CMS): See PAL package type 0020. The difference being that the package available is an asymmetric key package [RFC5958] that is signed and encapsulated in a signed data content type, as specified in Section 4.4.2 of [RFC7030]. Also, see Section 8.1 of this document.

0022 Asymmetric Key Package (PKCS#12): See PAL package type 0020. The difference being that the package available is PKCS12 [RFC7292] content type. See Section 8.3 of this document.

0023 Asymmetric Key Package Receipt or Error: Indicates that the server wants the client to return a key package receipt or error [RFC7191] to the /serverkeygen/return URI, which is defined in Section 8.

0024 Symmetric Key Package: Indicates that a symmetric key package [RFC6031] is available for the client by pointing to a /symmetrickeys URI, which is defined in Section 5.

0025 Symmetric Key Package Receipt or Error: Indicates that the server wants the client to return a key package receipt or an error [RFC7191] to the /symmetrickeys/return URI, which is defined in Section 5.

0026 Firmware Package: Indicates that a firmware package [RFC4108] is available for the client using the /firmware URI, which is defined in Section 6.

0027 Firmware Package Receipt or Error: Indicates that the server wants the client to return a firmware package load receipt or error [RFC4108] to the /firmware/return URI, which is defined in Section 6.

NOTE: The /tamp and tamp/return URIs are defined in Section 7.

0028 TAMP Status Query: Indicates that a TAMP Status Query package [RFC5934] is available for the client using the /tamp URI.

0029 TAMP Status Query Response or Error: Indicates that the server wants the client to return a TAMP Status Query Response or Error [RFC5934] to the /tamp/return URI.

0030 Trust Anchor Update: Indicates that a Trust Anchor Update package [RFC5934] is available for the client using the /tamp URI.

0031 Trust Anchor Update Confirm or Error: Indicates that the server wants the client to return a Trust Anchor Update Confirm or Error [RFC5934] to the /tamp/return URI.

0032 Apex Trust Anchor Update: Indicates that an Apex Trust Anchor Update package [RFC5934] is available for the client using the /tamp URI.

0033 Apex Trust Anchor Update Confirm or Error: Indicates that the server wants the client to return an Apex Trust Anchor Update Confirm or Error [RFC5934] to the /tamp/return URI.

0034 Community Update: Indicates that a Community Update package [RFC5934] is available for the client using the /tamp URI.

0035 Community Update Confirm or Error: Indicates that the server wants the client to return a Community Update Confirm or Error [RFC5934] to the /tamp/return URI.

0036 Sequence Number Adjust: Indicates that a Sequence Number Adjust package [RFC5934] is available for the client using the /tamp URI.

0037 Sequence Number Adjust Confirm or Error: Indicates that the server wants the client to return a Sequence Number Adjust Confirm or Error [RFC5934] to the /tamp/return URI.

2.1.2. PAL XML Schema

The name space is specified in Section 11.1. The fields in the schema were discussed earlier in Sections 2.1 and 2.1.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:pal="urn:ietf:params:xml:ns:pal"
  targetNamespace="urn:ietf:params:xml:ns:pal"
  elementFormDefault="qualified" attributeFormDefault="unqualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      This schema defines the types and elements needed
      to retrieve client packages from the server or for the
      client to post packages to the server.
    </xsd:documentation>
  </xsd:annotation>

  <!-- ===== Element Declarations ===== -->

  <xsd:element name="pal" type="pal:PAL" />

  <!-- ===== Complex Data Element Type Definitions ===== -->

  <xsd:complexType name="PAL">
    <xsd:annotation>
      <xsd:documentation>
        This type defines the Package Availability List (PAL).
```

```
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="message" type="pal:PALEntry"
          minOccurs="0" maxOccurs="unbounded">
          <xsd:annotation>
            <xsd:documentation>
              Contains information about the package and a link that
              the client uses to download or post the package.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="PALEntry">
      <xsd:annotation>
        <xsd:documentation>
          This type defines a product in the PAL.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="type" type="pal:PackageType" />
        <xsd:element name="date" type="pal:GeneralizedTimeType"
          minOccurs="0" />
        <xsd:element name="size" type="xsd:nonNegativeInteger">
          <xsd:annotation>
            <xsd:documentation>
              Indicates the package's size.
            </xsd:documentation>
          </xsd:annotation>
        <xsd:element name="info" type="pal:PackageInfoType" />
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="PackageInfoType">
      <xsd:annotation>
        <xsd:documentation>
          This type allows a choice of X.500 Distinguished Name,
          Subject Key Identifier, Issuer and Serial Number tuple,
          or URI.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:choice>
        <xsd:element name="dn"   type="pal:DistinguishedName" />
        <xsd:element name="ski"  type="pal:SubjectKeyIdentifier" />
        <xsd:element name="iasn" type="pal:IssuerAndSerialNumber" />
        <xsd:element name="uri"  type="pal:ThisURI" />
```

```
      </xsd:choice>
    </xsd:complexType>

    <xsd:complexType name="IssuerAndSerialNumber">
      <xsd:annotation>
        <xsd:documentation>
          This type holds the issuer Distinguished Name and
          serial number of a referenced certificate.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="issuer" type="pal:DistinguishedName" />
        <xsd:element name="serial" type="xsd:positiveInteger" />
      </xsd:sequence>
    </xsd:complexType>

    <!-- =====Simple Data Element Type Definitions ===== -->

    <xsd:simpleType name="PackageType">
      <xsd:annotation>
        <xsd:documentation>
          Identifies each package that a client may retrieve from
          the server with a 4-digit string.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="d{4}" />
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="GeneralizedTimeType">
      <xsd:annotation>
        <xsd:documentation>
          Indicates the date and time (YYYY-MM-DDTHH:MM:SSZ) the
          client last acknowledged successful receipt of the
          package or is absent if a) there is no indication
          the package has been downloaded or b) the PAL entry
          corresponds to a pointer to the next PAL.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:dateTime">
        <xsd:pattern value=".*:d{2}Z" />
        <xsd:minInclusive value="2013-05-23T00:00:00Z" />
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="DistinguishedName">
      <xsd:annotation>
```

```
        <xsd:documentation>
          This type holds an X.500 Distinguished Name.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:string">
        <xsd:maxLength value="1024" />
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="SubjectKeyIdentifier">
      <xsd:annotation>
        <xsd:documentation>
          This type holds a hex string representing the value of a
          certificate's SubjectKeyIdentifier.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:hexBinary">
        <xsd:maxLength value="1024" />
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="ThisURI">
      <xsd:annotation>
        <xsd:documentation>
          This type holds a URI, but is length limited.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:anyURI" />
      <xsd:maxLength value="1024" />
    </xsd:simpleType>

  </xsd:schema>
```

2.1.3. PAL JSON Object

The following is an example PAL JSON object. The fields in the object were discussed earlier in Sections 2.1 and 2.1.1.

```
[
  {
    "type": "0003",
    "date": "2016-12-29T09:28:00Z",
    "size": 1234,
    "info":
     {
      "uri": "https://www.example.com/.well-known/est/eecerts/1234"
     }
  },
```

```
    {
      "type": "0006",
      "date": "2016-12-29T09:28:00Z",
      "size": 1234,
      "info":
       {
        "iasn":
         {
          "issuer": "CN=Sean Turner,O=sn3rd,C=US",
          "serial": 0
         }
       }
    }
  ]
```

2.2. Request PAL

Clients request their PAL with an HTTP GET [RFC7231] using an operation path of "/pal". Clients indicate whether they would prefer XML or JSON by including the HTTP Accept header [RFC7231] with either "application/xml" or "application/json", respectively.

2.3. Provide PAL

If the server has a PAL for the client, the server response MUST contain an HTTP 200 response code with a content-type of "application/xml" [RFC7303] or "application/json" [RFC7159].

When the server constructs a PAL, an order of precedence for PAL offerings is based on the following rationale:

- /cacerts and /crls packages are the most important because they support validation decisions on certificates used to sign and encrypt other listed PAL items.

- /csrattrs are the next in importance, since they provide information that the server would like the client to include in its certificate enrollment request.

- /simpleenroll, /simplereenroll, and /fullcmc packages items are next in importance, since they can impact a certificate used by the client to sign CMS content or a certificate to establish keys for encrypting content exchanged with the client.

  * A client engaged in a certificate management SHOULD accept and process CA-provided transactions as soon as possible to avoid undue delays that might lead to protocol failure.

o /symmetrickeys, /firmware, /tamp, and /eecerts packages containing keys and other types of products are last. Precedence SHOULD be given to packages that the client has not previously downloaded. The items listed in a PAL may not identify all of the packages available for a device. This can be for any of the following reasons:

* The server may temporarily withhold some outstanding PAL items to simplify client processing.

* If a CA has more than one certificate ready for the client, the server will provide a notice for one at a time. Pending notices will be serviced in order of the earliest date when the certificate will be used.

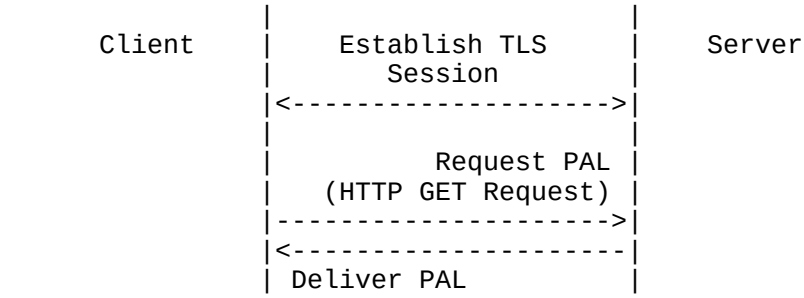
When rejecting a request the server specifies either an HTTP 4xx error, or an HTTP 5xx error.

All other return codes are handled as specified in Section 4.2.3 of [RFC7030] (i.e., 202 handling and all other HTTP response codes).

3. Distribute EE Certificates

Numerous mechanisms exist for clients to query repositories for certificates. The service provided by the /eecerts PC is different in that it is not a general purpose query for client certificates instead it allows the server to provide peer certificates to a client that the server knows through an out-of-band mechanism that the client will be communicating with. For example, a router being provisioned that connects to two peers can be provisioned with not only its certificate but also with the peers' certificates.

The server need not authenticate or authorize the client for distributing an EE certificate because the package contents are already signed by a CA (i.e., the certificate(s) in a certs-only message have already been signed by a CA). The message flow is similar to Figure 1 except that the connection need not be HTTPS:



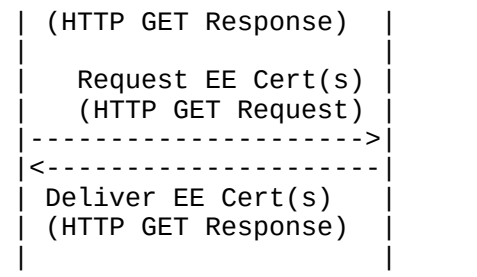


Figure 2 - /eecerts Message Sequence

3.1. EE Certificate Request

Clients request EE certificates with an HTTP GET [RFC7231] using an operation path of "/eecerts".

3.2. EE Certificate Response

The response and processing of the returned error codes is identical to that in Section 4.1.3 of [RFC7030] except that the certificate provided is not the one issued to the client but instead one or more client's peer certificates is returned in the certs-only message.

Clients MUST reject EE certificates that do not validate to an authorized TA.

4. Distribute CRLs and ARLs

CRLs (and ARLs) are needed in many instances to perform certificate path validation [RFC5280]. They can be obtained from repositories if their location is provided in the certificate. However, the client needs to parse the certificate and perform an additional round trip to retrieve them. Providing CRLs at the time of bootstrap obviates the need for the client to parse certificate and aid those clients who might be unable to retrieve the CRL. Clients are free to obtain CRLs on which they rely from sources other than the server (e.g., a local directory). The /crls PC allows servers to distribute CRLs at the same time clients retrieve their certificate(s) and CA certificate(s) as well as peer certificates.

The server need not authenticate or authorize the client for distributing a CRL because the package content is already signed by a CA (i.e., the CRLs in a crls-only message have already been signed by a CA). The message flow is as depicted in Figure 2 but with "CRL(s)" instead of "EE Cert(s)".

4.1. CRL Request

Clients request CRLs with an HTTP GET [RFC7231] using an operation path of "/crls".

4.2. CRL Response

The response and processing of the response is identical to that in Section 4.1.3 of [RFC7030] except that instead of providing the issued certificate one of more CRLs are returned in the crls-only message.

Clients MUST reject CRLs that do not validate to an authorized TA.

5. Symmetric Keys, Receipts, and Errors

In addition to public keys, clients often need one or more symmetric keys to communicate with their peers. The /symmetrickeys PC allows the server to distribute symmetric keys to clients.

Distribution of keys does not always work as planned and clients need a way to inform the server that something has gone wrong; they also need a way to inform the server, if asked, that the distribution process has successfully completed. The /symmetrickeys/return PC allows client to provide errors and receipts.

Clients MUST authenticate the server and clients MUST check the server’s authorization.

The server MUST authenticate clients and the server MUST check the client’s authorization.

HTTP GET [RFC7231] is used when the server provides the key to the client (see Section 5.1) using the /symmetrickeys PC; HTTP POST [RFC7231] is used when the client provides a receipt (see Section 5.2) or an error (see Section 5.2) to the server with the /symmetrickeys/return PC.

5.1. Symmetric Keys

Servers use /symmetrickeys to provide clients symmetric keys; symmetric key package is defined in [RFC6031].

As with the /serverkeygen PC defined in [RFC7030], the default distribution method of the symmetric key uses the encryption mode of the negotiated TLS cipher suite. Keys are not protected by preferred key wrapping methods such as AES Key Wrap [RFC3394] or AES Key Wrap with Padding [RFC5649] because encryption of the symmetric key beyond that provided by TLS is OPTIONAL. Therefore, the cipher suite used to return the symmetric key MUST offer commensurate cryptographic

strength with the symmetric key being delivered to the client. The cipher suite used MUST NOT have NULL encryption algorithm as this will disclose the unprotected symmetric key. It is strongly RECOMMENDED that servers always return encrypted symmetric keys.

The following depicts the protocol flow:

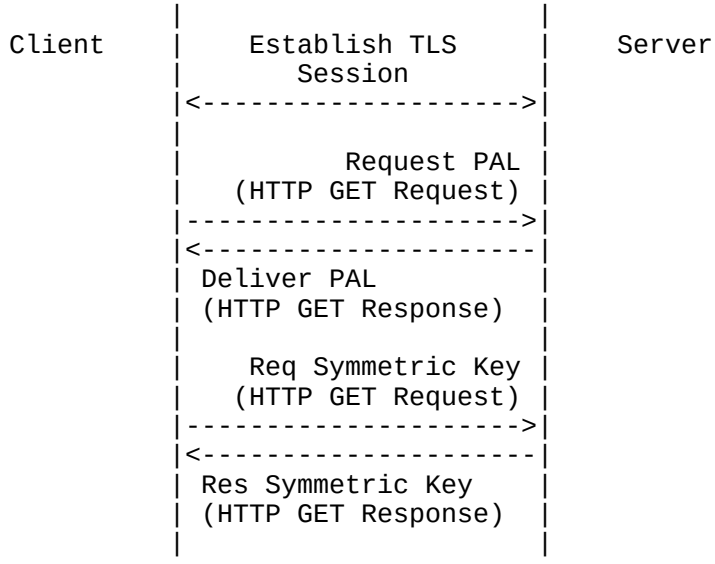


Figure 3 - /symmetrickeys Message Sequence

5.1.1. Distribute Symmetric Keys

Clients request the symmetric key from the server with an HTTP GET [RFC7231] using an operation path of "/symmetrickeys".

5.1.2. Symmetric Key Response

If the request is successful, the server response MUST have an HTTP 200 response code with a Content-Type of application/cms [RFC7193]. The optional application/cms encapsulatingContent and innerContent parameters SHOULD be included with the Content-Type to indicate the protection afforded to the returned symmetric key. The returned content varies:

- If additional encryption is not being employed, the content associated with application/cms is a DER-encoded [X.690] symmetric key package.

- If additional encryption is employed, the content associated with

application/cms is DER-encoded enveloped data that encapsulates a signed data that further encapsulates a symmetric key package.

- o If additional encryption and origin authentication are employed, the content associated with application/cms is a DER-encoded signed data that encapsulates an enveloped data that encapsulates a signed data that further encapsulates a symmetric key package.

- o If CCC (CMS Content Constraints) [RFC6010] is supported the content associated with application/cms is a DER-encoded encrypted key package [RFC6032]. Encrypted key package provides three choices to encapsulate keys: encrypted data, enveloped data, and authenticated enveloped data. Prior to employing one of these three encryption choices the key package can be encapsulated in a signed data.

How the server knows whether the client supports the encrypted key package is beyond the scope of this document.

When rejecting a request, the server specifies either an HTTP 4xx error, or an HTTP 5xx error.

If a symmetric key package (which might be signed) or an encrypted key package (which might be signed before and after encryption) is digitally signed, the client MUST reject it if the digital signature does not validate back to an authorized TA.

Note: absent a policy on the client side requiring signature, a malicious EST server can simply strip the signature, thus bypassing that check. In that case, this requirement is merely a sanity check, serving to detect mis-signed packages or misconfigured clients.

[RFC3370], [RFC5753], [RFC5754], [RFC6033], [RFC6160], and [RFC6161] provide algorithm details for use when protecting the symmetric key package and encrypted key package.

5.2. Symmetric Key Receipts and Errors

Clients use /symmetrickeys/return to provide symmetric key package receipts; the key package receipt content type is defined in [RFC7191]. Clients can be configured to automatically return receipts after processing a symmetric key package, return receipts based on processing of the key-package-identifier-and-receipt-request attribute [RFC7191], or return receipts when prompted by a PAL entry.

Servers can indicate that clients return a receipt by including the key-package-identifier-and-receipt-request attribute in a signed data as a signed attribute. However, this attribute only appears when

additional encryption is employed (see Section 5.1.2).

Clients also use /symmetrickeys/return to return symmetric key package errors; the key package error content type is defined in [RFC7191]. Clients can be configured to automatically return errors after processing a symmetric key package or based on a PAL entry.

The following depicts the protocol flow:

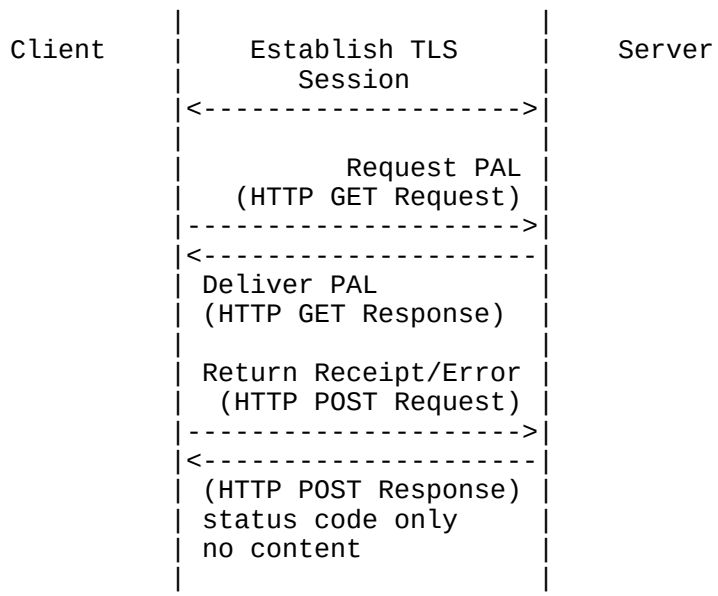


Figure 4 - /symmetrickeys/return Message Sequence

5.2.1. Provide Symmetric Key Receipt or Error

Clients return symmetric key receipts and errors to the server with an HTTP POST [RFC7231] using an operation path of "/symmetrickeys/return". The returned content varies:

- The key package receipt is digitally signed [RFC7191], the Content-Type is application/cms [RFC7193] and the associated content is signed data, which encapsulates a key package receipt.

- If the key package error is not digitally signed, the Content-Type is application/cms and the associated content is key package error. If the key package error is digitally signed, the Content-Type is application/cms and the associated content is signed data, which encapsulates a key package error.

The optional application/cms encapsulatingContent and innerContent parameters SHOULD be included with the Content-Type to indicate the protection afforded to the receipt or error.

[RFC3370], [RFC5753], [RFC5754], and [RFC7192] provide algorithm details for use when protecting the key package receipt or key package error.

5.2.2. Symmetric Key Receipt or Error Response

If the client successfully provides a receipt or error, the server response has an HTTP 204 response code (i.e., no content is returned).

When rejecting a request, the server specifies either an HTTP 4xx error, or an HTTP 5xx error.

If a key package receipt or key package error is digitally signed, the server MUST reject it if the digital signature does not validate back to an authorized TA.

6. Firmware, Receipts, and Errors

Servers can distribute object code for cryptographic algorithms and software with the firmware package [RFC4108].

Clients MUST authenticate the server and clients MUST check the server’s authorization.

The server MUST authenticate the client and the server MUST check the client’s authorization.

The /firmware PC uses an HTTP GET [RFC7231] and the /firmware/return PC uses an HTTP POST [RFC7231]. GET is used when the client retrieves firmware from the server (see Section 6.1); POST is used when the client provides a receipt (see Section 6.2) or an error (see Section 6.2).

6.1. Firmware

The /firmware URI is used by servers to provide firmware packages to clients.

The message flow is as depicted in Figure 3 modulo replacing "Symmetric Key" with "Firmware Package".

6.1.1. Distribute Firmware

Clients request firmware from the server with an HTTP GET [RFC7231] using an operation path of "/firmware".

6.1.2. Firmware Response

If the request is successful, the server response MUST have an HTTP 200 response code with a Content-Type of "application/cms" [RFC7193]. The optional encapsulatingContent and innerContent parameters SHOULD be included with Content-Type to indicate the protection afforded to the returned firmware. The returned content varies:

- If the firmware is unprotected, then the Content-Type is application/cms and the content is the DER-encoded [X.690] firmware package.
- If the firmware is compressed, then the Content-Type is application/cms and the content is the DER-encoded [X.690] compressed data that encapsulates the firmware package.
- If the firmware is encrypted, then the Content-Type is application/cms and the content is the DER-encoded [X.690] encrypted data that encapsulates the firmware package (which might be compressed prior to encryption).
- If the firmware is signed, then the Content-Type is application/cms and the content is the DER-encoded [X.690] signed data that encapsulates the firmware package (which might be compressed, encrypted, or compressed and then encrypted prior to signature).

How the server knows whether the client supports the unprotected, signed, compressed and/or encrypted firmware package is beyond the scope of this document

When rejecting a request, the server specifies either an HTTP 4xx error, or an HTTP 5xx error.

If a firmware package is digitally signed, the client MUST reject it if the digital signature does not validate back to an authorized TA.

[RFC3370], [RFC5753], and [RFC5754] provide algorithm details for use when protecting the firmware package.

6.2. Firmware Receipts and Errors

Clients use the /firmware/return PC to provide firmware package load receipts and errors [RFC4108]. Clients can be configured to automatically return receipts and errors after processing a firmware

package or based on a PAL entry.

The message flow is as depicted in Figure 4 modulo the receipt or error is for a firmware package.

6.2.1. Provide Firmware Receipt or Error

Clients return firmware receipts and errors to the server with an HTTP POST [RFC7231] using an operation path of "/firmware/return". The optional encapsulatingContent and innerContent parameters SHOULD be included with Content-Type to indicate the protection afforded to the returned firmware receipt or error. The returned content varies:

- If the firmware receipt is not digitally signed, the Content-Type is application/cms [RFC7193] and the content is the DER-encoded firmware receipt.
- If the firmware receipt is digitally signed, the Content-Type is application/cms and the content is the DER-encoded signed data encapsulating the firmware receipt.
- If the firmware error is not digitally signed, the Content-Type is application/cms and the content is the DER-encoded firmware error.
- If the firmware error is digitally signed, the Content-Type is application/cms and the content is the DER-encoded signed data encapsulating the firmware error.

[RFC3370], [RFC5753], and [RFC5754] provide algorithm details for use when protecting the firmware receipt or firmware error.

6.2.2. Firmware Receipt or Error Response

If the request is successful, the server response MUST have an HTTP 204 response code (i.e., no content is returned).

When rejecting a request, the server MUST specify either an HTTP 4xx error, or an HTTP 5xx error.

If a firmware receipt or firmware error is digitally signed, the server MUST reject it if the digital signature does not validate back to an authorized TA.

7. Trust Anchor Management Protocol

Servers distribute TAMP packages to manage TAs in a client’s trust anchor databases; TAMP packages are defined in [RFC5934]. TAMP will

allow the flexibility for a device to load authorities while maintaining an operational state. Unlike other systems that require new software loads when new PKI Roots are authorized for use, TAMP allows for automated management of roots for provisioning or replacement as needed.

Clients MUST authenticate the server and clients MUST check the server's authorization.

The server MUST authenticate the client and the server MUST check the client's authorization.

The /tamp PC uses an HTTP GET [RFC7231] and the tamp/return PC uses an HTTP POST [RFC7231]. GET is used when the server requests that the client retrieve a TAMP package (see Section 7.1); POST is used when the client provides a confirm (see Section 7.2), provides a response (see Section 7.2), or provides an error (see Section 7.2) for the TAMP package.

7.1. TAMP Status Query, Trust Anchor Update, Apex Trust Anchor Update, Community Update, and Sequence Number Adjust

Clients use the /tamp PC to retrieve the TAMP packages: TAMP Status Query, Trust Anchor Update, Apex Trust Anchor Update, Community Update, and Sequence Number Adjust. Clients can be configured to periodically poll the server for these packages or contact the server based on a PAL entry.

The message flow is as depicted in Figure 3 modulo replacing "Symmetric Key" with the appropriate TAMP message.

7.1.1. Request TAMP Packages

Clients request the TAMP packages from the server with an HTTP GET [RFC7231] using an operation path of "/tamp".

7.1.2. Return TAMP Packages

If the request is successful, the server response MUST have an HTTP 200 response code and a Content-Type of:

- application/tamp-status-query for TAMP Status Query
- application/tamp-update for Trust Anchor Update
- application/tamp-apex-update for Apex Trust Anchor Update
- application/tamp-community-update for Community Update
- application/tamp-sequence-adjust for Sequence Number Adjust

As specified in [RFC5934], these content types are digitally signed and clients must support validating the packages directly signed by

TAs. For this specification, clients MUST support validation with a certificate and clients MUST reject it if the digital signature does not validate back to an authorized TA.

[RFC3370], [RFC5753], and [RFC5754] provide algorithm details for use when protecting the TAMP packages.

7.2. TAMP Response, Confirm, and Errors

Clients return the TAMP Status Query Response, Trust Anchor Update Confirm, Apex Trust Anchor Update Confirm, Community Update Confirm, Sequence Number Adjust Confirm, and TAMP Error to servers using the /tamp/return PC. Clients can be configured to automatically return responses, confirms, and errors after processing a TAMP package or based on a PAL entry.

The message flow is as depicted in Figure 4 modulo replacing "Receipt/Error" with the appropriate TAMP response, confirm, or error.

7.2.1. Provide TAMP Response, Confirm, or Error

Clients provide the TAMP responses, confirms, and errors to the server with an HTTP POST using an operation path of "/tamp/return". Content-Type is:

- o application/tamp-status-query-response for TAMP Status Query Response
- o application/tamp-update-confirm for Trust Anchor Update Confirm
- o application/tamp-apex-update-confirm for Apex Trust Anchor Update Confirm
- o application/tamp-community-update-confirm for Community Update Confirm
- o application/tamp-sequence-adjust-confirm for Sequence Number Adjust Confirm
- o application/tamp-error for TAMP Error

As specified in [RFC5934], these content types should be signed. If signed, a signed data encapsulates the TAMP content.

[RFC3370], [RFC5753], and [RFC5754] provide algorithm details for use when protecting the TAMP packages.

7.2.2. TAMP Response, Confirm, and Error Response

If the request is successful, the server response MUST have an HTTP 204 response code (i.e., no content is returned).

When rejecting a request, the server MUST specify either an HTTP 4xx error, or an HTTP 5xx error.

If the package is digitally signed, the server MUST reject it if digital signature does not validate back to an authorized TA.

8. Asymmetric Keys, Receipts, and Errors

[RFC7030] defines the /serverkeygen PC to support server-side generation of asymmetric keys. Keys are returned either as an unprotected PKCS#8 when additional security beyond TLS is not employed or as a CMS asymmetric key package content type that is encapsulated in a signed data content type that is further encapsulated in an enveloped data content type when additional security beyond TLS is requested. Some implementations prefer the use of other CMS content types to encapsulate the asymmetric key package; this document extends the content types that can be returned in Section 8.1.

[RFC7191] defines content types for key package receipts and errors. This document defines the /serverkeygen/return PC to add support for returning receipts and errors for asymmetric key packages in Section 8.2.

PKCS#12 [RFC7292], sometimes referred to as "PFX" (Personal inFormation eXchange), "P12", and "PKCS#12" files, are often used to distribute asymmetric private keys and the associated certificate. This document extends the /serverkeygen PC to allow servers to distribute using PKCS#12 server-generated asymmetric private keys and the associated certificate to clients in Section 8.3.

8.1. Asymmetric Key Encapsulation

CMS supports a number of content types to encapsulate other CMS content types; [RFC7030] includes one such possibility; note that when only relying on TLS the returned key is not a CMS content type. This document extends the CMS content types that can be returned.

If the client supports CCC [RFC6010], then the client can indicate that it supports encapsulated asymmetric keys in the encrypted key package [RFC5958] by including the encrypted key package’s OID in a content type attribute [RFC2985] in the CSR (Certificate Signing Request), aka the certification request, it provides to the server. If the client knows a priori that the server supports the encrypted key package content type, then the client need not include the content type attribute in the CSR.

In all instances defined herein, the Content-Type is

"application/cms" [RFC7193]. The optional encapsulatingContent and innerContent parameters SHOULD be included with Content-Type to indicate the protection afforded to the returned asymmetric key package.

If additional encryption and origin authentication is employed, the content associated with application/cms is a DER-encoded signed data that encapsulates an enveloped data that encapsulates a signed data that further encapsulates an asymmetric key package.

If CCC (CMS Content Constraints) is supported and additional encryption is employed, the content associated with application/cms is a DER-encoded encrypted key package [RFC6032] content type that encapsulates a signed data that further encapsulates an asymmetric key package.

If CCC is supported and additional encryption and additional origin authentication is employed, the content associated with application/cms is a DER-encoded signed data that encapsulates an encrypted key package content type that encapsulates a signed data that further encapsulates an asymmetric key package.

Encrypted key package [RFC6032] provides three choices to encapsulate keys, encrypted data, enveloped data, and authenticated data, with enveloped data being the mandatory to implement choice.

When rejecting a request, the server specifies either an HTTP 4xx error, or an HTTP 5xx error.

If an asymmetric key package or an encrypted key package is digitally signed, the client MUST reject it if the digital signature does not validate back to an authorized TA.

Note: absent a policy on the client side requiring signature, a malicious EST server can simply strip the signature, thus bypassing that check. In that case, this requirement is merely a sanity check, serving to detect mis-signed packages or misconfigured clients.

[RFC3370], [RFC5753], [RFC5754], [RFC6033], [RFC6161], and [RFC6162] provide algorithm details for use when protecting the asymmetric key package and encrypted key package.

8.2. Asymmetric Key Package Receipts and Errors

Clients can be configured to automatically return receipts after processing an asymmetric key package, return receipts based on processing of the key-package-identifier-and-receipt-request attribute [RFC7191], or return receipts when prompted by a PAL entry.

Servers can indicate that clients return a receipt by including the key-package-identifier-and-receipt-request attribute [RFC7191] in a signed data as a signed attribute.

The protocol flow is identical to that depicted in Figure 4 modulo the receipt or error is for asymmetric keys.

The server and client processing is as described in Section 5.2.1 and 5.2.2 modulo the PC, which for Asymmetric Key Packages is "/serverkeygen/return".

8.3. PKCS#12

PFX is widely deployed and supports protecting keys in the same fashion as CMS but it does so differently.

8.3.1. Server-Side Key Generation Request

Similar to the other server-generated asymmetric keys provided through the /serverkeygen PC:

- The certificate request is HTTPS POSTed and is the same format as for the "/simpleenroll" and "/simplereenroll" path extensions with the same content-type and transfer encoding.

- In all respects, the server SHOULD treat the CSR as it would any enroll or re-enroll CSR; the only distinction here is that the server MUST ignore the public key values and signature in the CSR. These are included in the request only to allow re-use of existing codebases for generating and parsing such requests.

PBE (password based encryption) shrouding of PKCS#12 is supported and this specification makes no attempt to alter this de facto standard. As such, there is no support of the DecryptKeyIdentifier specified in [RFC7030] for use with PKCS#12 (i.e., "enveloping" is not supported). NOTE: Use of PBE requires the password be distributed to the client; methods to distribute this password are out-of-scope.

8.3.2. Server-Side Key Generation Response

If the request is successful, the server response MUST have an HTTP 200 response code with a content-type of "application/pkcs12" that consists of a base64-encoded DER-encoded [X.690] PFX [RFC7292].

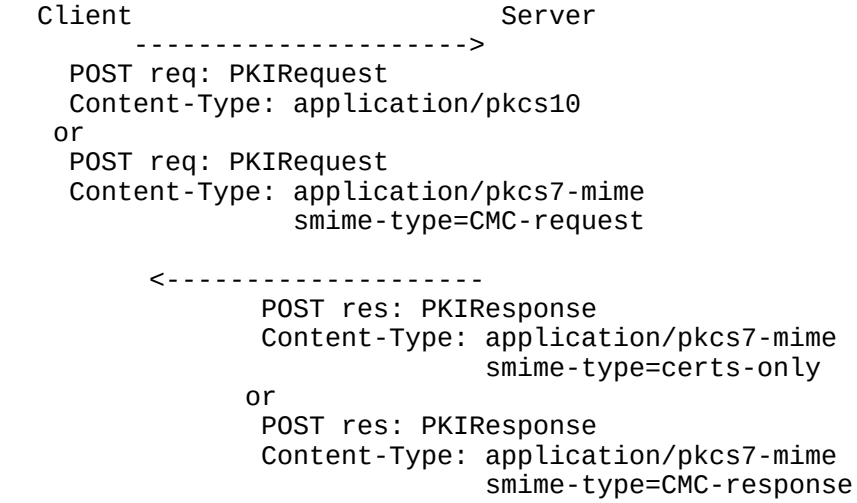
Note that this response is different than the response returned in Section 4.4.2 of [RFC7030] because here the private key and the certificate are included in the same PFX.

When rejecting a request, the server MUST specify either an HTTP 4xx error or an HTTP 5xx error. The response data's content-type MAY be "text/plain" [RFC2046] to convey human-readable error messages.

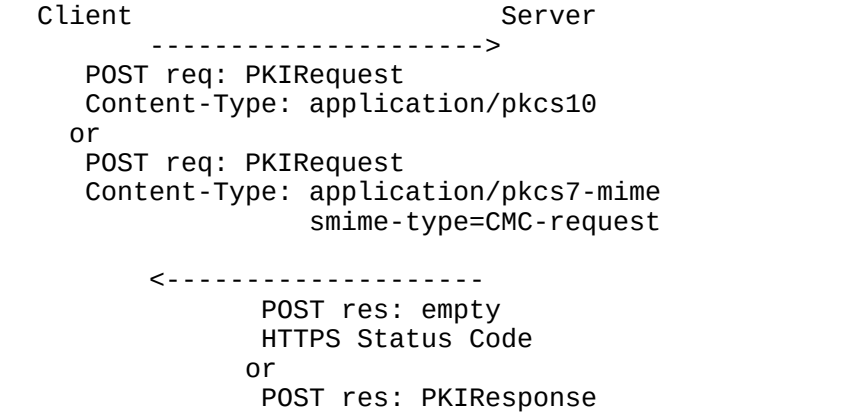
9. PAL & Certificate Enrollment

The /fullcmc PC is defined in [RFC7030]; the CMC (Certificate Management over Cryptographic Message Syntax) requirements and packages are defined in [RFC5272], [RFC5273], [RFC5274], and [RFC6402]. This section describes PAL interactions.

Under normal circumstances the client-server interactions for PKI enrollment are as follows:



if the response is rejected during the same session:



```
                        Content-Type: application/pkcs7-mime
                                      smime-type=CMC-response
```

if the request is to be filled later:

```
        Client                        Server
              ---------------------->
          POST req: PKIRequest
          Content-Type: application/pkcs10
         or
          POST req: PKIRequest
          Content-Type: application/pkcs7-mime
                        smime-type=CMC-request

              <----------------------
                     POST res: empty
                     HTTPS Status Code
                     + Retry-After
                    or
                     POST res: PKIResponse (pending)
                     Content-Type: application/pkcs7-mime
                                   smime-type=CMC-response

              ---------------------->
          POST req: PKIRequest (same request)
          Content-Type: application/pkcs10
         or
          POST req: PKIRequest (CMC Status Info only)
          Content-Type: application/pkcs7-mime
                        smime-type=CMC-request

              <----------------------
                     POST res: PKIResponse
                     Content-Type: application/pkcs7-mime
                                   smime-type=certs-only
                    or
                     POST res: PKIResponse
                     Content-Type: application/pkcs7-mime
                                   smime-type=CMC-response
```

With the PAL, the client begins after pulling the PAL and a Start Issuance PAL package type essentially adding the following before the request:

```
        Client                        Server
             ---------------------->
          GET req: PAL
```

```
                <--------------------
                       GET res: PAL
                       Content-Type: application/xml
```

The client then proceeds as above with a simple PKI Enroll, Full CMC Enrollment, or begin enrollment assisted with a CSR:

```
        Client                       Server
             ---------------------->
          GET req: DS certificate with CSR

                <--------------------
                       GET res: PAL
                       Content-Type: application/csr-attrs
```

For immediately rejected request, CMC works well. If the server prematurely closes the connection, then the procedures in Section 8.2.4 of [RFC7231] apply. But, this might leave the client and server in a different state. The client could merely resubmit the request but another option, documented herein, is for the client to instead download the PAL to see if the server has processed the request. Clients might also use this process when they are unable to remain connected to the server for the entire enrollment process; if the server does not or is not able to return a PKIData indicating a status of pending, then the client will not know whether the request was received. If a client uses the PAL and reconnects to determine if the certification or rekey or renew request was processed:

- Clients MUST authenticate the server and clients MUST check the server’s authorization.
- Server MUST authenticate the client and the server MUST check the client’s authorization.
- Clients retrieve the PAL using the /pal URI.
- Clients and servers use the operation path of "/simpleenroll", "simplereenroll", or "/fullcmc", based on the PAL entry, with an HTTP GET [RFC7231] to get the success or failure response.

Responses are as specified in [RFC7030].

10. Security Considerations

This document relies on many other specifications; however, all of the security considerations [RFC7030] apply. For HTTP, HTTPS, and TLS security considerations see [RFC7231], [RFC2818], and [RFC5246]; for URI security considerations see [RFC3986]; for content type

security considerations see [RFC4073], [RFC4108], [RFC5272], [RFC5652], [RFC5751], [RFC5934], [RFC5958] [RFC6031], [RFC6032], [RFC6268], [RFC6402], [RFC7191], and [RFC7292]; for algorithms used to protect packages see [RFC3370], [RFC5649], [RFC5753], [RFC5754], [RFC5959], [RFC6033], [RFC6160], [RFC6161], [RFC6162] and [RFC7192]; for random numbers see [RFC4086]; for server-generated asymmetric key pairs see [RFC7030].

11. IANA Considerations

IANA is requested to create the PAL Package Type registry and perform three registrations: PAL Name Space, PAL XML Schema, and PAL Package Types.

11.1. PAL Name Space

This section registers a new XML namespace [XMLNS], "urn:ietf:params:xml:ns:pal" per the guidelines in [RFC3688]:

URI: urn:ietf:params:xml:ns:pal
Registrant Contact: Sean Turner (sean@sn3rd.com)
XML:
BEGIN

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
   <title>Package Availability List</title>
</head>
<body>
   <h1>Namespace for Package Availability List</h1>
   <h2>urn:ietf:params:xml:ns:pal</h2>
   <p>See RFC TBD</p>
</body>
</html>
```

END

11.2. PAL XML Schema

This section registers an XML schema as per the guidelines in [RFC3688].

URI: urn:ietf:params:xml:schema:pal

Registrant Contact: Sean Turner sean@sn3rd.com

XML: See Section 2.1.2.

11.3. PAL Package Types

IANA is kindly requested to create a new registry named: PAL Package Type. This registry is for PAL Package Types whose initial values are found in Section 2.1.1. Future PAL Package Types registrations are to be subject to Expert Review, as defined in RFC 8126 [RFC8126]. Package types MUST be paired with a media type; package types specify the path component to be used that in turn specify the media type used.

12. Acknowledgements

Thanks in no particular order go to Alexey Melnikov, Paul Hoffman, Brad McInnis, Max Pritikin, Francois Rousseau, Chris Bonatti, and Russ Housley for taking time to provide comments.

13. References

13.1. Normative References

[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>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <http://www.rfc-editor.org/info/rfc2046>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2585] Housley, R. and P. Hoffman, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", RFC 2585, DOI 10.17487/RFC2585, May 1999, <http://www.rfc-editor.org/info/rfc2585>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, DOI 10.17487/RFC2985, November 2000, <http://www.rfc-editor.org/info/rfc2985>.

[RFC3370] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, DOI 10.17487/RFC3370, August 2002, <http://www.rfc-editor.org/info/rfc3370>.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002, <http://www.rfc-editor.org/info/rfc3394>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4073] Housley, R., "Protecting Multiple Contents with the Cryptographic Message Syntax (CMS)", RFC 4073, DOI 10.17487/RFC4073, May 2005, <http://www.rfc-editor.org/info/rfc4073>.

[RFC4108] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108, DOI 10.17487/RFC4108, August 2005, <http://www.rfc-editor.org/info/rfc4108>.

[RFC4514] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, DOI 10.17487/RFC4514, June 2006, <http://www.rfc-editor.org/info/rfc4514>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <http://www.rfc-editor.org/info/rfc5272>.

[RFC5273] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC): Transport Protocols", RFC 5273, DOI 10.17487/RFC5273, June 2008, <http://www.rfc-editor.org/info/rfc5273>.

[RFC5274] Schaad, J. and M. Myers, "Certificate Management Messages over CMS (CMC): Compliance Requirements", RFC 5274, DOI 10.17487/RFC5274, June 2008, <http://www.rfc-

editor.org/info/rfc5274>.

[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, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, DOI 10.17487/RFC5649, September 2009, <http://www.rfc-editor.org/info/rfc5649>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <http://www.rfc-editor.org/info/rfc5652>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <http://www.rfc-editor.org/info/rfc5751>.

[RFC5753] Turner, S. and D. Brown, "Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)", RFC 5753, DOI 10.17487/RFC5753, January 2010, <http://www.rfc-editor.org/info/rfc5753>.

[RFC5754] Turner, S., "Using SHA2 Algorithms with Cryptographic Message Syntax", RFC 5754, DOI 10.17487/RFC5754, January 2010, <http://www.rfc-editor.org/info/rfc5754>.

[RFC5934] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Management Protocol (TAMP)", RFC 5934, DOI 10.17487/RFC5934, August 2010, <http://www.rfc-editor.org/info/rfc5934>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <http://www.rfc-editor.org/info/rfc5958>.

[RFC5959] Turner, S., "Algorithms for Asymmetric Key Package Content Type", RFC 5959, DOI 10.17487/RFC5959, August 2010, <http://www.rfc-editor.org/info/rfc5959>.

[RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967, DOI 10.17487/RFC5967, August 2010, <http://www.rfc-editor.org/info/rfc5967>.

[RFC6010] Housley, R., Ashmore, S., and C. Wallace, "Cryptographic Message Syntax (CMS) Content Constraints Extension", RFC 6010, DOI 10.17487/RFC6010, September 2010, <http://www.rfc-editor.org/info/rfc6010>.

[RFC6031] Turner, S. and R. Housley, "Cryptographic Message Syntax (CMS) Symmetric Key Package Content Type", RFC 6031, DOI 10.17487/RFC6031, December 2010, <http://www.rfc-editor.org/info/rfc6031>.

[RFC6032] Turner, S. and R. Housley, "Cryptographic Message Syntax (CMS) Encrypted Key Package Content Type", RFC 6032, DOI 10.17487/RFC6032, December 2010, <http://www.rfc-editor.org/info/rfc6032>.

[RFC6033] Turner, S., "Algorithms for Cryptographic Message Syntax (CMS) Encrypted Key Package Content Type", RFC 6033, DOI 10.17487/RFC6033, December 2010, <http://www.rfc-editor.org/info/rfc6033>.

[RFC6160] Turner, S., "Algorithms for Cryptographic Message Syntax (CMS) Protection of Symmetric Key Package Content Types", RFC 6160, DOI 10.17487/RFC6160, April 2011, <http://www.rfc-editor.org/info/rfc6160>.

[RFC6161] Turner, S., "Elliptic Curve Algorithms for Cryptographic Message Syntax (CMS) Encrypted Key Package Content Type", RFC 6161, DOI 10.17487/RFC6161, April 2011, <http://www.rfc-editor.org/info/rfc6161>.

[RFC6162] Turner, S., "Elliptic Curve Algorithms for Cryptographic Message Syntax (CMS) Asymmetric Key Package Content Type", RFC 6162, DOI 10.17487/RFC6162, April 2011, <http://www.rfc-editor.org/info/rfc6162>.

[RFC6268] Schaad, J. and S. Turner, "Additional New ASN.1 Modules for the Cryptographic Message Syntax (CMS) and the Public Key Infrastructure Using X.509 (PKIX)", RFC 6268, DOI 10.17487/RFC6268, July 2011, <http://www.rfc-editor.org/info/rfc6268>.

[RFC6402] Schaad, J., "Certificate Management over CMS (CMC) Updates", RFC 6402, DOI 10.17487/RFC6402, November 2011, <http://www.rfc-editor.org/info/rfc6402>.

[RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <http://www.rfc-editor.org/info/rfc7303>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <http://www.rfc-editor.org/info/rfc7030>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7191] Housley, R., "Cryptographic Message Syntax (CMS) Key Package Receipt and Error Content Types", RFC 7191, DOI 10.17487/RFC7191, April 2014, <http://www.rfc-editor.org/info/rfc7191>.

[RFC7192] Turner, S., "Algorithms for Cryptographic Message Syntax (CMS) Key Package Receipt and Error Content Types", RFC 7192, DOI 10.17487/RFC7192, April 2014, <http://www.rfc-editor.org/info/rfc7192>.

[RFC7193] Turner, S., Housley, R., and J. Schaad, "The application/cms Media Type", RFC 7193, DOI 10.17487/RFC7193, April 2014, <http://www.rfc-editor.org/info/rfc7193>.

[RFC7231] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7292] Moriarty, K., Ed., Nystrom, M., Parkinson, S., Rusch, A., and M. Scott, "PKCS #12: Personal Information Exchange Syntax v1.1", RFC 7292, DOI 10.17487/RFC7292, July 2014, <http://www.rfc-editor.org/info/rfc7292>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <http://www.rfc-editor.org/info/rfc8126>.

[XML] W3C, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C Recommendation, November 2008, <http://www.w3.org/TR/2006/REC-xml-20060816/>.

[XMLSCHEMA] Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041082, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

[X.690] ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002. Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

13.2. Informative References

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, DOI 10.17487/RFC2985, November 2000, <http://www.rfc-editor.org/info/rfc2985>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

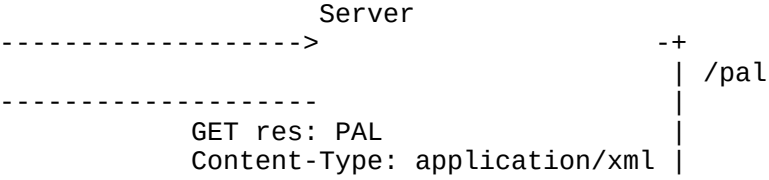
[XMLNS] Hollander, D., Bray, T., and A. Layman, "Namespaces in XML", World Wide Web Consortium First Edition REC-xml-names-19990114, January 1999, <http://www.w3.org/TR/1999/REC-xml-names-19990114>.

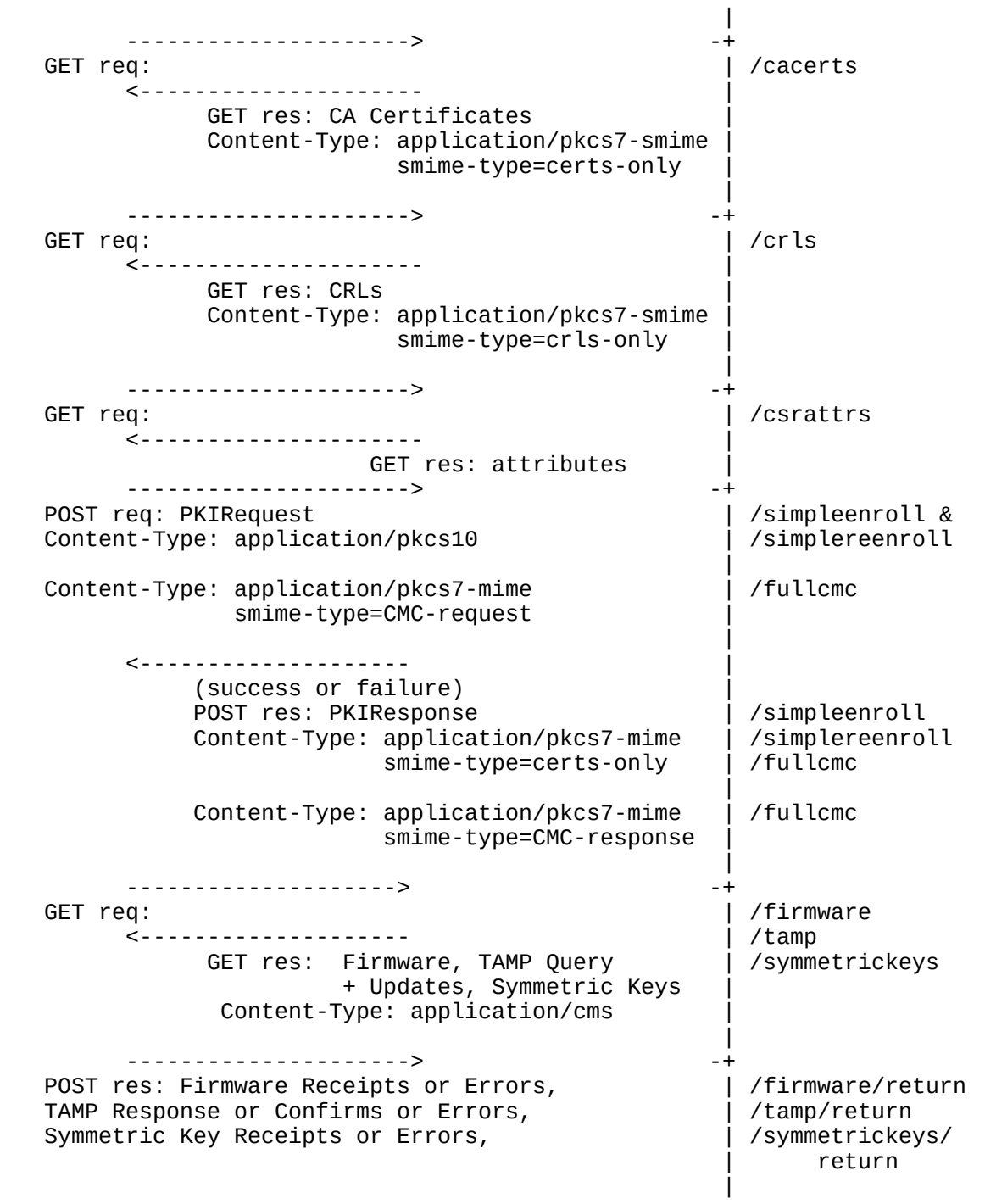
Appendix A. Example Use of PAL

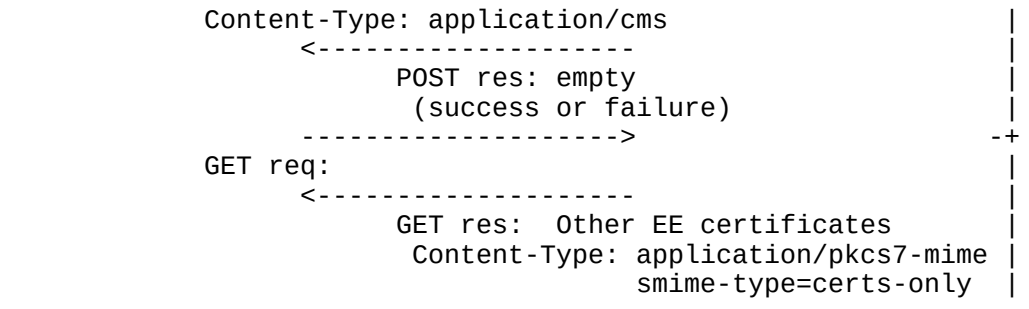
This is an informative appendix. It includes examples protocol flows.

Steps for using a PAL include:

1. Access PAL
2. Process PAL entries
   - 2.1. Get CA Certificates
   - 2.2. Get CRLs
   - 2.3. Get CSR attributes
   - 2.4. Enroll: simple enrollment, re-enrollment, or full CMC
   - 2.5. Get Firmware, TAMP, Symmetric Keys, or EE Certificates




GET req:
/cacerts
GET res: CA Certificates
Content-Type: application/pkcs7-smime
smime-type=certs-only
GET req:
/crls
GET res: CRLs
Content-Type: application/pkcs7-smime
smime-type=crls-only
GET req:
/csrattrs
GET res: attributes
POST req: PKIRequest
Content-Type: application/pkcs10
/simpleenroll &
/simplereenroll
Content-Type: application/pkcs7-mime
smime-type=CMC-request
/fullcmc
(success or failure)
POST res: PKIResponse
Content-Type: application/pkcs7-mime
smime-type=certs-only
/simpleenroll
/simplereenroll
/fullcmc
Content-Type: application/pkcs7-mime
smime-type=CMC-response
/fullcmc
GET req:
/firmware
/tamp
/symmetrickeys
GET res: Firmware, TAMP Query
+ Updates, Symmetric Keys
Content-Type: application/cms
POST res: Firmware Receipts or Errors,
TAMP Response or Confirms or Errors,
Symmetric Key Receipts or Errors,
/firmware/return
/tamp/return
/symmetrickeys/
return



The figure above shows /eecerts after /*/return, but this is for illustrative purposes only.

Appendix B. Additional CSR Attributes

This is an informative appendix.

In some cases, the client is severely limited in its ability to encode and decode ASN.1 objects. If the client knows a csr template is being provided during enrollment, then it can peel the returned csr attribute, generate its keys, place the public key in the certification request, and then sign the request. To accomplish this, the server returns a PKCS7PDU attribute [RFC2985] in as part of the /csrattrs (the following is pseudo ASN.1 and is only meant to show the fields needed to accomplish returning a template certification request):

```
pKCS7PDU ATTRIBUTE ::= {
  WITH SYNTAX ContentInfo
  ID pkcs-9-at-pkcs7PDU
  }

pkcs-9-at-pkcs7PDU OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  at(25) 5
  }
```

The ContentInfo is a PKIData:

```
PKIData ::= SEQUENCE {
  reqSequence        SEQUENCE SIZE(0..MAX) OF TaggedRequest,
  }
```

Where TaggedRequest is a choice between the PKCS #10 or CRMF requests.

```
TaggedRequest ::= CHOICE {
```

```
      tcr               [0] TaggedCertificationRequest,
      crm               [1] CertReqMsg,
      }
```

Or, the Content Info can be a signed data content type that further encapsulates a PKIData.

Authors' Addresses

Sean Turner
sn3rd

EMail: sean@sn3rd.com