Internet Engineering Task Force
Internet-Draft
Updates: 4210 (if approved)
Intended status: Standards Track
Expires: January 8, 2020

CMP Updates
draft-brockhaus-lamps-cmp-updates-00

Abstract

This document contains a set of updates to the base syntax of Certificate Management Protocol (CMP) version 2. This document updates RFC 4210.

Specifically, the CMP services updated in this document comprise:
Enable protection of server-side key generation using elliptic curve algorithms and the definition of an extended key usage to identify certificates of CMP endpoints on registration authorities and certification authorities.

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 January 8, 2020.

Copyright Notice

Copyright (c) 2019 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 ......................................................... 2
   1.1. Convention and Terminology ................................. 3
   2.1. New Section 1.1. - Changes since RFC 4210 ........... 3
   2.2. Replace Section 5.2.2. - Encrypted Values ............. 4
   2.3. Update Section 5.3.4. - Certification Response ........ 5
   2.4. Update Section 5.3.19.9. - Revocation Passphrase ..... 5
   2.5. Update Appendix B - The Use of Revocation Passphrase ... 6
   2.6. Update Appendix C - Request Message Behavioral Clarifications .......... 7
   2.7. Update Appendix D.4. - Request Message Behavioral Clarifications .......... 7
   2.8. Insert section for EKUs like kp-id-cmpRA definition at an appropriate location. .......... 7
3.  IANA Considerations ............................................. 7
4.  Security Considerations ........................................ 8
5.  Acknowledgements ................................................. 8
6.  References .......................................................... 8
   6.1. Normative References ........................................ 8
   6.2. Informative References ...................................... 9
Appendix A.  ASN.1 Modules ........................................ 9
Author's Address ...................................................... 9

1.  Introduction

While using CMP [RFC4210] in industrial and IoT environments and developing the Lightweight CMP Profile [I-D.brockhaus-lamps-industrial-cmp-profile] some limitations were identified in the original CMP specification. This document updates RFC 4210 [RFC4210] to overcome these limitations.

In general this document aims to improve the crypto agility of CMP to be flexible to react on future advances in cryptography.

This document also introduces a new extended key usage to identify CMP services on registration and certification authorities.

< TBD: While implementing CMP we identified some wording that could be more precise. It can be discussed if such wording issued need to be addressed in the context of this document or not. >

Brockhaus Expires January 8, 2020
1.1. Convention and Terminology

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

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

Technical terminology is used in conformance with RFC 4210 [RFC4210], RFC 4211 [RFC4211], and RFC 5280 [RFC5280]. The following key words are used:

CA: Certification authority, which issues certificates.

RA: Registration authority, an optional system component to which a CA delegates certificate management functions such as authorization checks.

EE: End entity, a user or device or service that holds a PKI certificate. An identifier for the EE is given as the subject of the certificate.


2.1. New Section 1.1. - Changes since RFC 4210

The following subsections describe feature updates to RFC 4210 [RFC4210]. They are always related to the base specification. Hence references to the original sections in RFC 4210 are used whenever possible.

Insert this section at the end of the current Section 1.

The following updates were made since RFC 4210:

- Offering envelopedData as another choice next to EncryptedValue to extend crypto agility in CMP. Note that according to RFC 4211 [RFC4211] section 2.1.9 the use of the EncryptedValue structure has been deprecated in favor of the EnvelopedData structure. For reasons of completeness and consistency the exchange of EncryptedValue with EncryptedKey is performed not only where required for the needed crypto agility for protection of centrally generated private key, but also for other purposes like encryption of revocation passphrases.
2.2. Replace Section 5.2.2. - Encrypted Values

Section 5.2.2 of RFC 4210 [RFC4210] describes the usage of EncryptedValue to transport encrypted data.

Replace the text of the section with the following text.

Where encrypted data (restricted, in this specification, to be either private keys, certificates or passwords) are sent in PKI messages, the EncryptedKey data structure is used.

EncryptedKey ::= CHOICE {
  encryptedValue        EncryptedValue, -- deprecated
  envelopedData     [0] EnvelopedData }

See CRMF [RFC4211] for EncryptedKey and EncryptedValue syntax.

Using this data structure, it offers the choice to either use EncryptedValue or EnvelopedData. As EncryptedValue offers only key transport, e.g. using RSA or symmetric encryption, EnvelopedData offers further key management techniques, e.g. key agreement, and more crypto agility. Note that according to RFC 4211 [RFC4211] section 2.1.9 the use of the EncryptedValue structure has been deprecated in favor of the EnvelopedData structure. Therefore, it is recommended to use EnvelopedData.

See CMS [RFC5652] for EnvelopedData syntax.

Using envelopedData within CMP contains only one recipientInfo structure because the content is encrypted only for one recipient.

Use of either EnvelopedData or EncryptedValue (for backward compatibility only) requires that the creator and intended recipient be able to encrypt and decrypt, respectively. Typically, this will mean that the sender and recipient have, or are able to generate, a shared secret key.

<TBD: Description of EnvelopedData structure parts which are used and filled to support application in CMP>
If EncryptedValue is used by the sender and the recipient of the PKIMessage already possesses a private key usable for decryption, then the encSymmKey field MAY contain a session key encrypted using the recipient’s public key.

2.3. Update Section 5.3.4. - Certification Response

Section 5.3.4 of RFC 4210 [RFC4210] describes the Certification Response. This document updates the syntax by using EncryptedKey instead of EncryptedValue as described in Section 2.1 above.

Replace the ASN.1 syntax of CertifiedKeyPair and CertOrEncCert with the following text.

CertifiedKeyPair ::= SEQUENCE {
  certOrEncCert       CertOrEncCert,
  privateKey      [0] EncryptedKey        OPTIONAL,
  -- see [CRMF] for comment on encoding
  publicationInfo [1] PKIPublicationInfo  OPTIONAL
}

CertOrEncCert ::= CHOICE {
  certificate     [0] Certificate,
  encryptedCert   [1] EncryptedKey
}

< Question to Jim Schaad: Simply exchanging EncryptedValue with EncryptedKey is what I understood from our discussion back in April. Is it completely backwards compatible to current ASN.1 Syntax of RFC4210? >

Add the following paragraphs to the end of the section.

In case EnvelopedData is the choice used for EncryptedKey, the encrypted private key or certificate MUST be placed in the envelopedData encryptedContentInfo encryptedContent OCTET STRING. Note that according to RFC 4211 [RFC4211] section 2.1.9 the use of the EncryptedValue structure has been deprecated in favor of the EnvelopedData structure.

2.4. Update Section 5.3.19.9. - Revocation Passphrase

Section 5.3.19.9 of RFC 4210 [RFC4210] describes the provisioning of a revocation passphrase for authenticating a later revocation request. This document updates the handling by using EncryptedKey instead of EncryptedValue to transport this information as described in Section 2.1 above.
Replace the text of the section with the following text.

The revocation passphrase MAY be used by the EE to send a passphrase to a CA/RA for the purpose of authenticating a later revocation request (in the case that the appropriate signing private key is no longer available to authenticate the request). See Appendix B for further details on the use of this mechanism.

GenMsg: {id-it 12}, EncryptedKey
GenRep: {id-it 12}, < absent >

In case EnvelopedData is the choice used for EncryptedKey, the encrypted revocation passphrase MUST be placed in the envelopedData encryptedContentInfo encryptedContent OCTET STRING. Note that according to RFC 4211 [RFC4211] section 2.1.9 the use of the EncryptedValue structure has been deprecated in favor of the EnvelopedData structure.

2.5. Update Appendix B - The Use of Revocation Passphrase

Appendix B of RFC 4210 [RFC4210] describes the usage of the revocation passphrases. As this document updates RFC 4210 [RFC4210] to utilize EncryptedKey in favor of EncryptedValue as described in Section 2.1 above, the description is updated accordingly.

Replace the first bullet point of this section with the following text.

- The OID and value specified in Section 5.3.19.9 of RFC 4210 [RFC4210] MAY be sent in a GenMsg message at any time, or MAY be sent in the generalInfo field of the PKIHeader of any PKIMessage at any time. (In particular, the EncryptedKey may be sent in the header of the certConf message that confirms acceptance of certificates requested in an initialization request or certificate request message.) This conveys a revocation passphrase chosen by the entity (i.e., and for use of EnvelopedData this is in the decrypted bytes of encryptedContent of the EnvelopedData structure and for use of EncryptedValue this is in the decrypted bytes of the encValue field) to the relevant CA/RA; furthermore, the transfer is accomplished with appropriate confidentiality characteristics. Note that according to RFC 4211 [RFC4211] section 2.1.9 the use of the EncryptedValue structure has been deprecated in favor of the EnvelopedData structure.

Replace the third bullet point of this section with the following text.
When using EnvelopedData the contentType of EncryptedContentInfo and when using EncryptedValue the valueHint field MAY contain a key identifier (chosen by the entity, along with the passphrase itself) to assist in later retrieval of the correct passphrase (e.g., when the revocation request is constructed by the entity and received by the CA/RA).

2.6. Update Appendix C - Request Message Behavioral Clarifications

Appendix C of [RFC4210] provides clarifications to the request message behavior. As this document updates [RFC4210] to utilize EncryptedKey in favor of EncryptedValue as described in Section 2.1 above, the description is updated accordingly.

Replace the note coming after the ASN.1 syntax of POPOPrivKey of this section with the following text.

```
-- **********
-- * the type of "thisMessage" is given as BIT STRING in RFC 4211
-- * [RFC4211]; it should be "EncryptedKey" (in accordance with
-- * Section 5.2.2, "Encrypted Values", of this specification).
-- * Therefore, this document makes the behavioral clarification of
-- * specifying that the contents of "thisMessage" MUST be encoded
-- * either as EnvelopedData or EncryptedValue (only for backward
-- * compatibility) and then wrapped in a BIT STRING. This allows
-- * the necessary conveyance and protection of the private key
-- * while maintaining bits-on-the-wire compatibility with RFC 4211
-- * [RFC4211].
-- **********
```

2.7. Update Appendix D.4. - Request Message Behavioral Clarifications

< TBD, add further details on crc[1].certifiedKeyPair, e.g. refer to usage of EncryptedValue protected with the pre-shared symmetric MACing key using SYM_PENC_ALG. >

2.8. Insert section for EKUs like kp-id-cmpRA definition at an appropriate location.

< Details need to be defined later >

3. IANA Considerations

<Add any IANA considerations>
4. Security Considerations

No changes are made to the existing security considerations of RFC 4210 [RFC4210].

5. Acknowledgements

We would like to thank the various reviewers of this document.

6. References

6.1. Normative References


6.2.  Informative References

[I-D.brockhaus-lamps-industrial-cmp-profile]

Appendix A.  ASN.1 Modules

Changes to the following parts are needed

- Import from PKIKXCRMF-2005

CertTemplate, PKIPublicationInfo, EncryptedKey, CertId,
CertReqMessages
FROM PKIXCRMF-2005 {iso(1) identified-organization(3)
dod(6) internet(1) security(5) mechanisms(5) pkix(7)
id-mod(0) id-mod-crmf2005(36)}

- In CertifiedKeyPair, CertOrEncCert and id-it-revPassphrase

CertifiedKeyPair ::= SEQUENCE {
certOrEncCert       CertOrEncCert,
privateKey      [0] EncryptedKey        OPTIONAL,
-- see [CRMF] for comment on encoding
publicationInfo [1] PKIPublicationInfo  OPTIONAL
}

CertOrEncCert ::= CHOICE {
certificate     [0] CMPCertificate,
encryptedCert   [1] EncryptedKey
}

--   id-it-revPassphrase    OBJECT IDENTIFIER ::= {id-it 12}
--      RevPassphraseValue      ::= EncryptedKey

< TBD: If needed the complete ASN.1 Module from RFC 4210 section needs to be copied here to be changed. >

Author’s Address
Hendrik Brockhaus
Siemens AG
Otto-Hahn-Rin 6
Munich  81739
Germany

Email: hendrik.brockhaus@siemens.com
URI:   http://www.siemens.com/
Abstract

The goal of this document is to facilitate interoperability and automation by profiling the Certificate Management Protocol (CMP) version 2 and the related Certificate Request Message Format (CRMF) version 2. It specifies a subset of CMP and CRMF focusing on typical uses cases relevant for managing certificates of devices in many industrial and IoT scenarios. To limit the overhead of certificate management for constrained devices only the most crucial types of transactions are specified as mandatory. To foster interoperability also in more complex scenarios, other types of transactions are specified as recommended or optional.

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 January 8, 2020.

Copyright Notice

Copyright (c) 2019 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.
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. History of changes ............................................. 3
2. Introduction .................................................... 4
  2.1. Motivation for profiling CMP ................................. 4
  2.2. Motivation for a lightweight profile for CMP .................. 5
  2.3. Existing CMP profiles ......................................... 6
  2.4. Compatibility with existing CMP profiles ...................... 6
  2.5. Scope of this document ....................................... 7
  2.6. Structure of this document .................................... 8
  2.7. Convention and Terminology ................................... 8
3. Architecture and use cases ....................................... 9
  3.1. Solution architecture ........................................ 9
  3.2. Basic generic CMP message content ............................ 10
  3.3. Supported use cases .......................................... 10
    3.3.1. Mandatory use cases ..................................... 11
    3.3.2. Recommended Use Cases .................................. 11
    3.3.3. Optional use cases ...................................... 12
  3.4. CMP message transport ....................................... 12
4. Generic parts of the PKI message .................................. 13
  4.1. General description of the CMP message header ................. 13
  4.2. General description of the CMP message protection ............ 15
  4.3. General description of CMP message extraCerts ................ 15
5. End Entity focused certificate management use cases ............... 16
  5.1. Requesting a new certificate from a PKI ....................... 16
    5.1.1. A certificate from a new PKI with signature protection ... 18
    5.1.2. Update an existing certificate with signature protection ... 23
    5.1.3. A certificate from a PKI with MAC protection ............. 24
    5.1.4. A certificate from a legacy PKI using PKCS#10 request ....... 26
    5.1.5. Generate the key pair centrally at the (L)RA/CA .......... 26
    5.1.6. Delayed enrollment .................................... 27
    5.1.7. Omitted confirmation ................................... 28
  5.2. Revoking a certificate ....................................... 28
  5.3. Error reporting ............................................. 30
  5.4. Support messages ........................................... 32
    5.4.1. Root CA certificate update ................................ 32
    5.4.2. Get enrollment voucher .................................. 32
6. LRA and RA focused certificate management use cases .............. 33
  6.1. Forwarding of messages ..................................... 33
6.1.1. Not changing protection ..................... 35
6.1.2. Replacing protection .......................... 35
  6.1.2.1. Keeping proof-of-possession ............... 36
  6.1.2.2. Breaking proof-of-possession .............. 36
6.1.3. Initiating delayed enrollment .................. 37
6.1.4. Granting omitted confirmation .................. 37
6.2. Revoking certificates on behalf of another’s entities .... 37
6.3. Error reporting .................................. 38
7. CMP message transport variants ....................... 38
  7.1. HTTP transport .................................. 38
  7.2. HTTPS transport using certificates ............... 39
  7.3. HTTPS transport using shared secrets .......... 39
  7.4. File-based transport ............................. 40
  7.5. CoAP transport .................................. 40
  7.6. Piggybacking on other reliable transport ....... 40
8. IANA Considerations ................................ 40
9. Security Considerations .............................. 40
10. Acknowledgements .................................. 40
11. References .................................. 41
  11.1. Normative References ......................... 41
  11.2. Informative References ....................... 41
Appendix A. Additional Stuff .......................... 43
Authors’ Addresses .................................. 43

1. History of changes

From version 00 -> 01:

- Change focus from industrial to more multi-purpose use cases and lightweight CMP profile.

- Incorporate the omitted confirmation into the header specified in section Section 4.1 and described in the standard enrollment use case in section Section 5.1.1 due to discussion with Tomas Gustavsson.

- Change from OPTIONAL to RECOMMENDED for use case ‘Revoke another’s entities certificate’ in section Section 6.2 and, because it is regarded as important functionality in many environments to enable the management station to revoke EE certificates.

- Complete the specification of the revocation message flow in section Section 5.2 and Section 6.2.

- The CoAP based transport mechanism and piggybacking of CMP messages on top of other reliable transport protocols is out of scope of this document and would need to be specified in another document.
2. Introduction

This document specifies certificate management transactions implementing machine-to-machine and IoT use cases. The focus lies on maximum automation and interoperable implementation of all involved components from end entities (EE) through an optional Local Registration Authority (LRA) and the RA up to the CA. The profile makes use of the concepts and syntax specified in CMP [RFC4210], CRMF [RFC4211], and HTTP transfer for CMP [RFC6712]. Especially CMP and CRMF are very feature-rich standards, while only a limited subset of the specified functionality is needed in many environments. Additionally, the standards are not always precise enough on how to interpret and implement the described concepts. Therefore, we aim at tailoring and specifying in more detail how to use these concepts to implement lightweight automated certificate management.

2.1. Motivation for profiling CMP

CMP was standardized in 1999 and is implemented in several CA products. In 2005 a completely reworked and enhanced version 2 of CMP [RFC4210] and CRMF [RFC4211] has been published followed by a document specifying a transfer mechanism using http [RFC6712] in 2012.

Though CMP is a very solid and capable protocol it could be used more widely. The most important reason for not more intense application of CMP appears to be that the protocol is offering a large set of features and options but being not always precise enough and leaving room for interpretation. On the one hand, this makes CMP applicable to a very wide range of scenarios, but on the other hand a full implementation of all options is unrealistic because this would take enormous effort.

Moreover, many details of the CMP protocol have been left open or have not been specified in full preciseness. The profiles specified in Appendix D and E of [RFC4210] offer some more detailed certificate use cases. But the specific needs of highly automated scenarios for a machine-to-machine communication are not covered sufficiently.

As also 3GPP, and UNISG already put across, profiling is a way of coping with the challenges mentioned above. To profile means to take advantage of the strengths of the given protocol, while explicitly narrowing down the options it provides to exactly those needed for the purpose(s) at hand and eliminating all identified ambiguities. In this way all the general and applicable aspects of the protocol
can be taken over and only the peculiarities of the target scenario need to be dealt with specifically.

Doing such a profiling for a new target environment can be a high effort because the range of available options needs to be well understood and the selected options need to be consistent with each other and with the intended usage scenario. Since most industrial use cases typically have much in common it is worth sharing this effort, which is the aim of this document. Other standardization bodies can then reference the profile from this document and do not need to come up with individual profiles.

2.2. Motivation for a lightweight profile for CMP

The profiles specified in Appendix D and E of CMP have been developed in particular to manage certificates of human end entities. With the evolution of distributed systems and client-server architectures, certificates for machines and applications on them have become widely used. This trend has strengthened even more in emerging industrial and IoT scenarios. CMP is sufficiently flexible to support these very well.

Today’s IT security architectures for industrial solutions typically use certificates for endpoint authentication within protocols like IPSec, TLS or SSH. Therefore, the security of these architectures highly relies upon the security and availability of the implemented certificate management procedures.

Due to increasing security in operational networks as well as availability requirements, especially on critical infrastructures and systems with a high volume of certificates, a state-of-the-art certificate management must be constantly available and cost-efficient, which calls for high automation and reliability. Such PKI operation according to commonly accepted best practices is also required in IEC 62443-3-3 [IEC62443-3-3] for security level 2 up to security level 4.

Further challenges in many industrial systems are network segmentation and asynchronous communication, where PKI operation is often not deployed on-site but in a more protected environment of a data center or trust center. Certificate management must be able to cope with such network architectures. CMP offers the required flexibility and functionality, namely self-contained messages, efficient polling, and support for asynchronous message transfer with end-to-end security.
2.3. Existing CMP profiles

As already stated, CMP contains profiles with mandatory and optional transactions in the Appendixes D and E of [RFC4210]. Those profiles focus on management of human user certificates and do not address the specific needs for certificate management automation for unattended machine or application-oriented end entities.


UNISIG has included a CMP profile for certificate enrollment in the subset 137 specifying the ETRAM/ECTS on-line key management for train control systems [UNISIG] in 2015.

Both standardization bodies use CMP [RFC4210], CRMF [RFC4211], and HTTP transfer for CMP [RFC6712] to add tailored means for automated certificate management for unattended machine or application-oriented end entities.

2.4. Compatibility with existing CMP profiles

The profile specified in this document is compatible with CMP [RFC4210] Appendixes D and E (PKI Management Message Profiles), with the following exceptions:

- signature-based protection is the default protection; initial transactions may also use HMAC,
- certification of a second key pair within the same transaction is not supported,
- proof-of-possession (POPO) with self-signature of the certTemplate according to [RFC4211] section 4.1 clause 3 is the only supported POPO method,
- confirmation of newly enrolled certificates may be omitted, and
- all transactions consist of request-response message pairs originating at the EE, i.e., announcement messages are omitted.

The profile specified in this document is compatible with the CMP profile for UMTS, LTE, and 5G network domain security and authentication framework [ETSI-3GPP], except that:
o protection of initial transactions may be HMAC-based,
o the subject name is mandatory in certificate templates, and
o confirmation of newly enrolled certificates may be omitted.

The profile specified in this document is compatible with the CMP profile for on-line key management in rail networks as specified in UNISIG subset-137 [UNISIG], except that:

o as of RFC 4210 [RFC4210] the messageTime is required to be Greenwich Mean Time coded as generalizedTime (Note: While UNISIG explicitely states that the messageTime in required to be 'UTC time', it is not clear if this means a coding as UTCTime or generalizedTime and if other time zones than Greenwich Mean Time shall be allowed. Therfore UNISG may be in conflict with RFC 4210 [RFC4210]. Both time formats are described in RFC 5280 [RFC5280] section 4.1.2.5.), and

o in case the request message is MAC protected, also the response, certConf, and PKIconf messages have a MAC-based protection (Note: if changing to signature protection of the response the caPubs field cannot be used securely anymore.).

2.5. Scope of this document

This document specifies requirements on generating messages on the sender side. It does not specify strictness of verification on the receiving side and how in detail to handle error cases.

Especially on the EE side this profile aims at a lightweight protocol that can be implemented on constrained devices. On the side of the central PKI components the profile accepts higher resource needs.

For the sake of robustness and preservation of security properties implementations should, as far as security is not affected, adhere to Postel’s law: "Be conservative in what you do, be liberal in what you accept from others" (often reworded as: "Be conservative in what you send, be liberal in what you accept").

When in chapter 3, 4, and 5 a field of the ASN.1 syntax as defined in RFC 4210 [RFC4210] and RFC 4211 [RFC4211] is not explicitly specified, it SHOULD not be used by the sending entity. The receiving entity MUST NOT require its absence and if present SHOULD ignore it.
2.6. Structure of this document

Chapter 2 introduces the general PKI architecture and approach to certificate management using CMP that is assumed in this document. Then it enlists the certificate management use cases specified in this document and describes them in general words. The list of supported certificate management use cases is divided into mandatory, recommended, and optional ones.

Chapter 3 profiles the CMP message header, protection, and extraCerts section as they are general elements of CMP messages.

Chapter 4 profiles the exchange of CMP messages between an EE and the first PKI component. There are various flavors of certificate enrollment requests optionally with polling, revocation, error handling, and general support transactions.

Chapter 5 profiles the exchange between further PKI components. These are in the first place the forwarding of messages coming from or going to an EE. This includes also initiating delayed delivery of messages, which involves polling. Additionally, it specifies transactions where the PKI component manages certificates on behalf of an EE or for itself.

Chapter 6 outlines different mechanisms for CMP message transfer, namely http-based transfer as already specified in [RFC6712], using an additional TLS layer, offline file-based transport, CoAP [RFC7252], or piggybacking CMP messages on other protocols.

2.7. Convention and Terminology

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

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

Technical terminology is used in conformance with RFC 4210 [RFC4210], RFC 4211 [RFC4211], RFC 5280 [RFC5280], and IEEE 802.1AR [IEEE802.1AR]. The following key words are used:

**CA:** Certification authority, which issues certificates.

**RA:** Registration authority, an optional system component to which a CA delegates certificate management functions such as authorization checks.
3. Architecture and use cases

3.1. Solution architecture

Typically, a machine EE will be equipped with a manufacturer issued certificate during production. Such a manufacturer issued certificate is installed during production to identify the device throughout its lifetime. This manufacturer certificate can be used to protect the initial enrollment of operational certificates after installation of the EE in a plant or industrial network. An operational certificate is issued by the owner or operator of the device to identify the device during operation, e.g., within a security protocol like IPSec, TLS, or SSH. In IEEE 802.1AR [IEEE802.1AR] a manufacturer certificate is called IDevID certificate and an operational certificate is called LDevID certificate.

All certificate management transactions are initiated by the EE. The EE creates a CMP request message, protects it using its manufacturer or operational certificate, if available, and sends it to its locally reachable PKI component. This PKI component may be an LRA, RA, or the CA, which checks the request, responds to it itself, or forwards the request upstream to the next PKI component. In case an (L)RA changes the CMP request message header or body or wants to prove a successful verification or authorization, it can apply a protection of its own. Especially the communication between an LRA and RA can be performed synchronously or asynchronously. Synchronous communication describes a timely uninterrupted communication between two communication partners, as asynchronous communication is not performed in a timely consistent manner, e.g., because of a delayed message delivery.
In operation environments a layered LRA-RA-CA architecture can be deployed, e.g., with LRAs bundling requests from multiple EEs at dedicated locations and one (or more than one) central RA aggregating the requests from multiple LRAs. Every (L)RA in this scenario will have its own dedicated certificate and private key allowing it to protect CMP messages it processes (CMP signing key/certificate). The figure above shows an architecture using one LRA and one RA. It is also possible to have only an RA or multiple LRAs and/or RAs. Depending on the network infrastructure, the communication between different PKI components may be synchronous online-communication, delayed asynchronous communication, or even offline file transfer.

Third-party CAs typically implement different variants of CMP or even use proprietary interfaces for certificate management. Therefore, the LRA or the RA may need to adapt the exchanged CMP messages to the flavor of communication required by the CA.

3.2. Basic generic CMP message content

Section 4 specifies the generic parts of the CMP messages as used later in Section 5 and Section 6.

- Header of a CMP message; see Section 4.1.
- Protection of a CMP message; see Section 4.2.
- ExtraCert field of a CMP message; see Section 4.3.

3.3. Supported use cases

Following the outlined scope from Section 2.5, this section gives a brief overview of the certificate management use cases specified in Section 5 and Section 6 and points out, if a implementation by compliant EE or PKI component is mandatory, recommended or optional.
3.3.1. Mandatory use cases

The mandatory use case in this document shall limit the overhead of certificate management for constrained devices to the most crucial types of transactions.

Section 5 - End Entity focused certificate management use cases

o Request a certificate from a new PKI with signature protection; see Section 5.1.1.

o Request to update an existing certificate with signature protection; see Section 5.1.2.

o Error reporting; see Section 5.3.

Section 6 - LRA and RA focused certificate management use cases

o Forward messages without changes; see Section 6.1.1.

o Forward messages with replaced protection and raVerified as proof-of-possession; see Section 6.1.2.2.

o Error reporting; see Section 6.3.

3.3.2. Recommended Use Cases

Additional recommended use cases shall support some more complex scenarios, that are considered as beneficial for environments with more specific boundary conditions.

Section 5 - End Entity focused certificate management use cases

o Request a certificate from a PKI with MAC protection; see Section 5.1.3.

o Handle delayed enrollment due to asynchronous message delivery.

< Motivation see Section 5.1.6, specification TBD >

o Revoke an own certificate.

Section 6 - LRA and RA focused certificate management use cases

o Revoke another’s entities certificate.
3.3.3. Optional use cases

The optional use cases support specific requirements seen only in a subset of environments.

Section 5 - End Entity focused certificate management use cases

o Request a certificate from a legacy PKI using a PKCS#10 [RFC2986] request.
  < Motivation see Section 5.1.4, specification TBD >

o Add central generation of a key pair to a certificate request.
  < Motivation see Section 5.1.5, specification TBD >

o Additional support messages, e.g., to update a Root CA certificate or to request an RFC 8366 [RFC8366] voucher.
  < Motivation see Section 5.4, specification TBD >

Section 6 - LRA and RA focused certificate management use cases

o Initiate delayed enrollment due to asynchronous message delivery.
  < Motivation see Section 6.1.3, specification TBD >

3.4. CMP message transport

Recommended transport

o Transfer CMP messages using HTTP; see Section 7.1.

Optional transport

o Transfer CMP messages using HTTPS with certificate-based authentication; see Section 7.2.

o Transfer CMP messages using HTTPS with shared-secret based protection; see Section 7.3.

o File-based CMP message transport.
  < Motivation see Section 7.4, specification TBD >
4. Generic parts of the PKI message

To reduce redundancy in the text and to ease implementation, the contents of the header, protection, and extraCerts fields of the CMP messages used in the transactions specified in Section 5 and Section 6 are standardized to the maximum extent possible. Therefore, the generic parts of a CMP message are described centrally in this section.

As described in section 5.1 of [RFC4210], all CMP messages have the following general structure:

```
+--------------------------------------------+
| PKIMessage                                 |
| +----------------------------------------+ |
| | header                                 | |
| +----------------------------------------+ |
| +----------------------------------------+ |
| | body                                   | |
| +----------------------------------------+ |
| +----------------------------------------+ |
| | protection (OPTIONAL)                  | |
| +----------------------------------------+ |
| +----------------------------------------+ |
| | extraCerts (OPTIONAL)                  | |
| +----------------------------------------+ |
+--------------------------------------------+
```

Figure 2: CMP message structure

The general contents of the message header, protection, and extraCerts fields are specified in the Section 4.1 to Section 4.3.

In case a specific CMP message needs different contents in the header, protection, or extraCerts fields, the differences are described in the respective message.

The CMP message body contains the message-specific information. It is described in the context of Section 5 and Section 6.

The behavior in case an error occurs while handling a CMP message is described in Section 6.3.

4.1. General description of the CMP message header

This section describes the generic header field of all CMP messages with signature-based protection. The only variations described here
are in the fields recipient, transactionID, and recipNonce of the first message of a transaction.

In case a message has MAC-based protection the changes are described in the respective section. The variations will affect the fields sender, protectionAlg, and senderKID.

For requirements about proper random number generation please refer to [RFC4086]. Any message-specific fields or variations are described in the respective sections of this chapter.

header
  pvno                        REQUIRED
    -- MUST be set to 2 to indicate CMP V2
  sender                      REQUIRED
    -- MUST be the subject of the signing certificate used for protection of this message
  recipient                   REQUIRED
    -- MUST be the name of the intended recipient
    -- If this is the first message of a transaction: SHOULD be the subject of the issuing CA certificate
    -- In all other messages: SHOULD be the same name as in the sender field of the previous message in this transaction
  messageTime                 RECOMMENDED
    -- MUST be the time at which the message was produced, if present
  protectionAlg               REQUIRED
    -- MUST be the algorithm identifier of the signature algorithm used for calculation of the protection bits
    -- The signature algorithm MUST be consistent with the SubjectPublicKeyInfo field of the signer’s certificate
    -- The hash algorithm used SHOULD be SHA-256
  algorithm                   REQUIRED
    -- MUST be the OID of the signature algorithm, like sha256WithRSAEncryption or ecdsa-with-SHA256
  parameters                  PROHIBITED
    -- MUST be absent
  senderKID                   RECOMMENDED
    -- MUST be the SubjectKeyIdentifier, if available, of the certificate used for protecting this message
  transactionID               REQUIRED
    -- If this is the first message of a transaction:
    -- MUST be 128 bits of random data for the start of a transaction to reduce the probability of having the transactionID already in use at the server
    -- In all other messages:
    -- MUST be the value from the previous message in the same transaction
senderNonce REQUIRED
-- MUST be fresh 128 random bits
recipNonce RECOMMENDED
-- If this is the first message of a transaction: SHOULD be
-- absent
-- In all other messages: MUST be present and contain the value
-- from senderNonce of the previous message in the same
-- transaction
generalInfo OPTIONAL
implicitConfirm OPTIONAL
ImplicitConfirmValue REQUIRED
-- The field is optional though it only applies to ir/cr/kur/p10cr
-- requests and ip/cp/kup responses
-- ImplicitConfirmValue of the request message MUST be NULL if
-- the EE wants to request not to send a confirmation message
-- ImplicitConfirmValue MUST be set to NULL if the (L)RA/CA wants
-- to grant not sending a confirmation message

4.2. General description of the CMP message protection

This section describes the generic protection field of all CMP
messages with signature-based protection.

protection REQUIRED
-- MUST contain the signature calculated using the signature
-- algorithm specified in protectionAlg

Only for MAC-based protection major differences apply as described in
the respective message.

The CMP message protection provides, if available, message origin
authentication and integrity protection for the CMP message header
and body. The CMP message extraCerts is not covered by this
protection.

NOTE: The requirements for checking certificates given in [RFC5280]
MUST be followed for the CMP message protection. OCSP or CRLs SHOULD
be used for status checking of the CMP signer certificates of
communication partners.

4.3. General description of CMP message extraCerts

This section describes the generic extraCerts field of all CMP
messages with signature-based protection.
extraCerts
-- SHOULD contain the signing certificate together with its
-- chain, if needed
-- If present, the first certificate in this field MUST
-- be the certificate used for signing this message
-- Self-signed certificates SHOULD NOT be included in
-- extraCerts and MUST NOT be trusted based on the listing in
-- extraCerts in any case

5. End Entity focused certificate management use cases

This chapter focuses on the communication of the EE and the first PKI
component it talks to. Depending on the network and PKI solution,
this will either be the LRA, the RA or the CA.

Profiles of the Certificate Management Protocol (CMP) [RFC4210]
handled in this chapter cover the following certificate management
use cases:

- Requesting a certificate from a PKI with variations like initial
  requests and updating, central key generation <TBD> and different
  protection means

- Revocation of a certificate <TBD>

- General messages for further support functions <TBD>

The use cases mainly specify the message body of the CMP messages and
utilize the specification of the message header, protection and
extraCerts as specified in Section 5.

The behavior in case an error occurs is described in Section 5.3.

This chapter is aligned to Appendix D and Appendix E of [RFC4210].
The general rules for interpretation stated in Appendix D.1 in
[RFC4210] need to be applied here, too.

This document does not mandate any specific supported algorithms like
Appendix D.2 of [RFC4210], [ETSI-3GPP], and [UNISIG] do. Using the
message sequences described here require agreement upon the
algorithms to support and thus the algorithm identifiers for the
specific target environment.

5.1. Requesting a new certificate from a PKI

There are different approaches to request a certificate from a PKI.
These approaches differ on the one hand in the way the EE can authenticate itself to the PKI it wishes to get a new certificate from and on the other hand in its capabilities to generate a proper new key pair. The authentication means may be as follows:

- Using a certificate from a trusted PKI and the corresponding private key, e.g., a manufacturer certificate
- Using the certificate to be updated and the corresponding private key
- Using a shared secret known to the EE and the PKI

Typically, such EE requests a certificate from a CA. When the (L)RA/CA responds with a message containing a certificate, the EE MUST reply with a confirmation message. The (L)RA/CA then MUST send confirmation back, closing the transaction.

The message sequences in this section allow the EE to request certification of a locally generated public-private key pair. (The functional extension for central key generation is TBD if needed. >) For requirements about proper random number and key generation please refer to [RFC4086]. The EE MUST provide a signature-based proof-of-possession of the private key associated with the public key contained in the certificate request as defined by [RFC4211] section 4.1 case 3. To this end it is assumed that the private key can technically be used as signing key. The most commonly used algorithms are RSA and ECDSA, which can technically be used for signature calculation regardless of potentially intended restrictions of the key usage.

The requesting EE provides the binding of the proof-of-possession to its identity by signature-based or MAC-based protection of the CMP request message containing that POPO. The (L)RA/CA needs to verify whether this EE is authorized to obtain a certificate with the requested subject and other attributes and extensions. Especially when removing the protection provided by the EE and applying a new protection the (L)RA MUST verify in particular the included proof-of-possession self-signature of the certTemplate using the public key of the requested certificate and MUST check that the EE, as authenticated by the message protection, is authorized to request a certificate with the subject as specified in the certTemplate (see Section 6.1.2).

There are several ways to install the Root CA certificate of a new PKI on an EE. The installation can be performed in an out-of-band manner, using a voucher [RFC8366] for enrollment, or by the caPubs field in the certificate response message. In case the installation
of the new Root CA certificate is performed using the caPubs field, the certificate response message MUST be properly authenticated, and the sender of this message MUST be authorized to install new Root CA certificates on the EE. This authorization MUST be indicated by the extended key usage in the (L)RA/CA certificate as specified in CMP Updates [brockhaus-lamps-cmp-updates].

5.1.1. A certificate from a new PKI with signature protection

This message sequence should be used by an EE to request a certificate of a new PKI using an existing certificate from an external PKI, e.g. a manufacturer certificate, to prove its identity to the new PKI. The EE already has established trust in this new PKI it is about to enroll to, e.g., by configuration means. The initialization request message is signature-protected using the existing certificate.

Preconditions:

1. The EE MUST have a certificate enrolled by an external PKI in advance to this transaction to authenticate itself to the (L)RA/CA using signature-based protection, e.g., using a manufacturer certificate.

2. The EE SHOULD know the subject name of the new CA it requests a certificate from; this name MAY be established using an enrollment voucher or other configuration means. If the EE does not know the name of the CA, the (L)RA/CA MUST know where to route this request to.

3. The EE MUST authenticate responses from the (L)RA/CA; trust MAY be established using an enrollment voucher or other configuration means.

4. The (L)RA/CA MUST trust the external PKI the EE uses to authenticate itself; trust MAY be established using some configuration means.

This message sequence is like that given in [RFC4210] Appendix E.7.
Message flow:

Step# EE                   (L)RA/CA
1   format ir             ->   ir   ->
2                      handle, re-protect or
3                                        forward ir
4                                        format or receive ip
5                                        possibly grant implicit
6                                        confirm
7                      <-   ip     <-
8   handle ip
9   format certConf (optional)  ->   certConf  ->
10                                        handle, re-protect or
11                                        forward certConf
12                                        format or receive PKIConf
13                      <-   pkiConf   <-
14   handle pkiConf (optional)

For this message sequence the EE MUST include exactly one single
CertReqMsg in the ir. If more certificates are required, further
requests MUST be sent using separate CMP Messages. If the EE wants
to omit sending a certificate confirmation message after receiving
the ip to reduce the number of protocol messages exchanged in a
transaction, it MUST request this by setting the implicitControlValue
in the ir to NULL.

If the CA accepts the request it MUST return the new certificate in
the certifiedKeyPair field of the ip message. If the EE requested to
omit sending a certConf message after receiving the ip, the (L)RA/CA
MAY confirm this by also setting the implicitControlValue in the ip
to NULL.

If the EE did not request implicit confirmation or the request was
not granted by the (L)RA/CA the confirmation as follows MUST be
performed. If the EE successfully receives the certificate and
accepts it, the EE MUST send a certConf message, which MUST be
answered by the (L)RA/CA with a pkiConf message. If the (L)RA/CA
does not receive the expected certConf message in time it MUST handle
this like a rejection by the EE.

If the certificate request was refused by the CA, the (L)RA/CA must
return an ip message containing the status code "rejection" and no
certifiedKeyPair field. Such an ip message MUST NOT be followed by the certConf and pkiConf messages.

Detailed message description:

Certification Request -- ir

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>As described in section 3.1</td>
</tr>
<tr>
<td>body</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The request of the EE for a new certificate</td>
</tr>
<tr>
<td>ir</td>
<td>REQUIRED</td>
</tr>
<tr>
<td></td>
<td>MUST be exactly one CertReqMsg</td>
</tr>
<tr>
<td></td>
<td>If more certificates are required, further requests MUST be</td>
</tr>
<tr>
<td></td>
<td>packaged in separate PKI Messages</td>
</tr>
<tr>
<td>certReq</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>certReqId</td>
<td>REQUIRED</td>
</tr>
<tr>
<td></td>
<td>MUST be set to 0</td>
</tr>
<tr>
<td>certTemplate</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>version</td>
<td>OPTIONAL</td>
</tr>
<tr>
<td>subject</td>
<td>REQUIRED</td>
</tr>
<tr>
<td></td>
<td>MUST contain the suggested subject name of the EE</td>
</tr>
<tr>
<td>certificate</td>
<td></td>
</tr>
<tr>
<td>publicKey</td>
<td>REQUIRED</td>
</tr>
<tr>
<td></td>
<td>MUST include the subject public key algorithm ID and value</td>
</tr>
<tr>
<td></td>
<td>extensions OPTIONAL</td>
</tr>
<tr>
<td></td>
<td>MAY include end-entity-specific X.509 extensions of the</td>
</tr>
<tr>
<td></td>
<td>requested certificate like subject alternative name,</td>
</tr>
<tr>
<td></td>
<td>key usage, and extended key usage</td>
</tr>
<tr>
<td>Popo</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>POPOSigningKey</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>poposkInput</td>
<td>PROHIBITED</td>
</tr>
<tr>
<td></td>
<td>MUST NOT be used because subject and publicKey are both</td>
</tr>
<tr>
<td></td>
<td>present in the certTemplate</td>
</tr>
<tr>
<td>algorithmIdentifier</td>
<td>REQUIRED</td>
</tr>
<tr>
<td></td>
<td>The signature algorithm MUST be consistent with the</td>
</tr>
<tr>
<td></td>
<td>publicKey field of the certTemplate</td>
</tr>
<tr>
<td></td>
<td>The hash algorithm used SHOULD be SHA-256</td>
</tr>
<tr>
<td>signature</td>
<td>REQUIRED</td>
</tr>
<tr>
<td></td>
<td>MUST be the signature computed over the DER-encoded</td>
</tr>
<tr>
<td></td>
<td>certTemplate</td>
</tr>
</tbody>
</table>

Brockhaus, et al. Expires January 8, 2020
extraCerts
   -- As described in section 3.3

Certification Response -- ip
Field                Value
header                
   -- As described in section 3.1

body
   -- The response of the CA to the request as appropriate
   ip                    REQUIRED
   caPubs                OPTIONAL
   -- MAY be used
   -- If used it MUST contain only the root certificate of the
   -- certificate contained in certOrEncCert
   response              REQUIRED
   -- MUST be exactly one CertResponse
   certReqId             REQUIRED
   -- MUST be set to 0
   status                REQUIRED
   -- PKIStatusInfo structure MUST be present
   status                REQUIRED
   -- positive values allowed: "accepted", "grantedWithMods"
   -- negative values allowed: "rejection"
   -- In case of rejection no certConf and pkiConf messages will
   -- be sent
   statusString          OPTIONAL
   -- MAY be any human-readable text for debugging, logging or to
   -- display in a GUI
   failInfo              OPTIONAL
   -- MUST be present if status is "rejection" and in this case
   -- the transaction MUST be terminated
   -- MUST be absent if the status is "accepted" or
   -- "grantedWithMods"
   certifiedKeyPair      OPTIONAL
   -- MUST be present if status is "accepted" or "grantedWithMods"
   -- MUST be absent if status is "rejection"
   certOrEncCert         REQUIRED
   -- MUST be present when certifiedKeyPair is present
   certificate           REQUIRED
   -- MUST be present when certifiedKeyPair is present
   -- MUST contain the newly enrolled X.509 certificate

protection
   -- As described in section 3.2
extraCerts  REQUIRED
  -- As described in section 3.3
  -- MUST contain the chain of the issued certificate
  -- Duplicate certificates MAY be omitted

Certificate Confirmation -- certConf

Field     Value

header
  -- As described in section 3.1

body
  -- The message of the EE sends confirmation to the (L)RA/CA
  -- to accept or reject the issued certificates
certConf  REQUIRED
  -- MUST be exactly one CertStatus
certStatus REQUIRED
  certHash  REQUIRED
  -- MUST be the hash of the certificate, using the same hash
  -- algorithm as used to create the certificate signature
certReqId REQUIRED
  -- MUST be set to 0
status    RECOMMENDED
  -- PKIStatusInfo structure SHOULD be present
  -- Omission indicates acceptance of the indicated certificate
status    REQUIRED
  -- positive values allowed: "accepted"
  -- negative values allowed: "rejection"
statusString OPTIONAL
  -- MAY be any human-readable text for debugging or logging
failInfo  OPTIONAL
  -- MUST be present if status is "rejection"
  -- MUST be absent if the status is "accepted"

protection REQUIRED
  -- As described in section 3.2
  -- MUST use the same certificate as for protection of the ir
extraCerts RECOMMENDED
  -- SHOULD contain the protection certificate together with its
  -- chain
  -- If present, the first certificate in this field MUST be the
  -- certificate used for signing this message
  -- Self-signed certificates SHOULD NOT be included in
  -- extraCerts and
  -- MUST NOT be trusted based on the listing in extraCerts in
PKI Confirmation -- pkiConf

Field                         Value

header
   -- As described in section 3.1

body
   pkiConf                     REQUIRED
      -- The content of this field MUST be NULL

protection                    REQUIRED
   -- As described in section 3.2
   -- SHOULD use the same certificate as for protection of the ip

extraCerts                    RECOMMENDED
   -- SHOULD contain the protection certificate together with its
   -- chain
   -- If present, the first certificate in this field MUST be the
   -- certificate used for signing this message
   -- Self-signed certificates SHOULD NOT be included in extraCerts
   -- and
   -- MUST NOT be trusted based on the listing in extraCerts in
   -- any case

5.1.2. Update an existing certificate with signature protection

This message sequence should be used by an EE to request an update of
one of the certificates it already has and that is still valid. The
EE uses the certificate it wishes to update to prove its identity and
possession of the private key for the certificate to be updated to
the PKI. Therefore, the key update request message is signed using
the certificate that is to be updated.

The general message flow for this message sequence is the same as
given in Section 5.1.1.

Preconditions:

1 The certificate the EE wishes to update MUST NOT be expired or
   revoked.

2 A new public-private key pair SHOULD be used.
The message sequence for this exchange is like that given in [RFC4210] Appendix D.6.

The message sequence for this exchange is identical to that given in Section 5.1.1, with the following changes:

1 The body of the first request and response MUST be kur and kup, respectively.

2 Protection of the kur MUST be performed using the certificate to be updated.

3 The subject field of the CertTemplate MUST contain the subject name of the existing certificate to be updated, without modifications.

4 The CertTemplate MUST contain the subject, issuer and publicKey fields only.

5 The regCtrl OldCertId SHOULD be used to make clear, even in case an (L)RA changes the message protection, which certificate is to be.

6 The caPubs field in the kup message MUST be absent.

As part of the certReq structure of the kur the control is added right after the certTemplate.

controls
  type                    RECOMMENDED
  -- MUST be the value id-regCtrl-oldCertID, if present
  value
  issuer                  REQUIRED
  serialNumber            REQUIRED
  -- MUST contain the issuer and serialNumber of the certificate
  -- to be updated

5.1.3. A certificate from a PKI with MAC protection

This message sequence should be used by an EE to request a certificate of a new PKI without having a certificate to prove its identity to the target PKI, but there is a shared secret established between the EE and the PKI. Therefore, the initialization request is MAC-protected using this shared secret. The (L)RA checking the MAC-protection SHOULD replace this protection according to Section 6.1.2 in case the next hop does not know the shared secret too.
For requirements with regard to proper random number and key generation please refer to [RFC4086].

The general message flow for this message sequence is the same as given in Section 5.1.1.

Preconditions:

1. The EE and the (L)RA/CA MUST share a symmetric key, this MAY be established by a service technician during initial local configuration.

2. The EE SHOULD know the subject name of the new CA it requests a certificate from; this name MAY be established using an enrollment voucher or other configuration means. If the EE does not know the name of the CA, the (L)RA/CA MUST know where to route this request to.

3. The EE MUST authenticate responses from the (L)RA/CA; trust MAY be established using the shared symmetric key.

The message sequence for this exchange is like that given in [RFC4210] Appendix D.4.

The message sequence for this exchange is identical to that given in Section 5.1.1, with the following changes:

1. The protection of all messages MUST be calculated using Message Authentication Code (MAC); the protectionAlg field MUST be id-PasswordBasedMac as described in section 5.1.3.1 of [RFC4210].

2. The sender MUST contain a name representing the originator of the message. The senderKID MUST contain a reference all participating entities can use to identify the symmetric key used for the protection.

3. The extraCerts of the ir, certConf, and PKICConf messages MUST be absent.

4. The extraCerts of the ip message MUST contain the chain of the issued certificate and root certificates SHOULD not be included and MUST NOT be trusted in any case.

Part of the protectionAlg structure, where the algorithm identifier MUST be id-PasswordBasedMac, is a PBMPParameter sequence. The fields of PBMPParameter SHOULD remain constant throughout this certificate management transaction to reduce the computational overhead.
PBMParameter REQUIRED
salt REQUIRED
-- MUST be the random value to salt the secret key
owf REQUIRED
-- MUST be the algorithm identifier for the one-way function
-- used
-- The one-way function SHA-1 MUST be supported due to
-- [RFC4211] requirements, but SHOULD NOT be used any more
-- SHA-256 SHOULD be used instead
iterationCount REQUIRED,
-- MUST be a limited number of times the OWF is applied
-- To prevent brute force and dictionary attacks a reasonable
-- high number SHOULD be used
mac REQUIRED
-- MUST be the algorithm identifier of the MAC algorithm used
-- The MAC function HMAC-SHA1 MUST be supported due to
-- [RFC4211] requirements, but SHOULD NOT be used any more
-- HMAC-SHA-256 SHOULD be used instead

5.1.4. A certificate from a legacy PKI using PKCS#10 request

This message sequence should be used by an EE to request a certificate of a legacy PKI only capable to process PKCS#10 [RFC2986] certification requests. The EE can prove its identity to the target PKI by using various protection means as described in Section 5.1.1 or Section 5.1.3.

In contrast to the other transactions described in Section 5.1, this transaction uses PKCS#10 [RFC2986] instead of CRMF [RFC4211] for the certificate request for compatibility reasons with legacy CA systems that require a PKCS#10 certificate request and cannot process CMP [RFC4210] or CRMF [RFC4211] messages. In such case the (L)RA can extract the PKCS#10 certificate request from the p10cr and provide it separately to the CA.

< Details need to be defined later >

5.1.5. Generate the key pair centrally at the (L)RA/CA

It is strongly preferable to generate public-private key pairs locally at the EE. Together with proof-of-possession of the private key in the certification request, this is to make sure that only the entity identified in the newly issued certificate has the private key.

There are some rare cases where an EE is not able or not willing to locally generate the new key pair. Reasons for this may be the following:
Lack of sufficient initial entropy.

Note: Good random numbers are not only needed for key generation, but also for session keys and nonces in any security protocol. Therefore, we believe that a decent security architecture should anyway support good random number generation on the EE side or provide enough entropy for the RNG seed during manufacturing to guarantee good initial pseudo-random number generation.

Due to lack of computational resources, e.g., in case of RSA keys.

Note: As key generation can be performed in advance to the certificate enrollment communication, it is typical not time critical.

Note: Besides the initial enrollment right after the very first bootup of the device, where entropy available on the device may be insufficient, we do not see any good reason for central key generation.

As the protection of centrally generated keys in the response message is being extended from EncryptedValue to EncryptedKey by CMP Updates [brockhaus-lamps-cmp-updates] also the alternative EnvelopedData can be used. As EncryptedValue offers only key transport, e.g. using RSA or symmetric encryption, EnvelopedData offers further key management techniques, e.g. key agreement, and therefore more crypto agility.

Note that according to RFC 4211 [RFC4211] section 2.1.9 the use of the EncryptedValue structure has been deprecated in favor of the EnvelopedData structure.

< Details need to be defined later >

5.1.6. Delayed enrollment

This functional extension can be applied in combination with certificate enrollment as described in Section 5.1.1 to Section 5.1.4. The functional extension can be used in case a (L)RA/CA cannot respond to the certificate request in a timely manner, e.g. due to offline upstream communication or registration officer interaction. Depending on the PKI architecture, it is not necessarily the PKI component directly communicating with the EE that initiates the delayed enrollment. In this case this PKI component MUST include the status waiting in the response and this response MUST not contain a newly issued certificate. When receiving a response with status waiting the EE MUST send a poll request to the (L)RA/CA. The (L)RA/CA MUST answers with a poll response containing a checkAfter time. This value indicates the minimum number of
seconds that must elapse before the EE sends another poll request. As soon as the (L)RA/CA can provide the final response message for the initial request of the EE, it MUST provide this in response to a poll request. After receiving this response, the EE can continue the original message sequence as described in the respective section of this document, e.g. send a certConf message.

< Details need to be defined later >

5.1.7. Omitted confirmation

This section will be removed though the functionality was incorporated into the header specified in section Section 4.1 and described in the standard enrollment use case in section Section 5.1.1 due to discussion with Tomas Gustavsson.

5.2. Revoking a certificate

This message sequence should be used by an entity to request the revocation of a certificate. Here the revocation request is used by an EE to revoke one of its own certificates. A (L)RA could also act as an EE to revoke one of its own certificates.

The revocation request message MUST be signed using the certificate that is to be revoked to prove the authorization to revoke to the PKI. The revocation request message is signature-protected using this certificate.

An EE requests the revocation of an own certificate at the CA that issued this certificate. The (L)RA/CA responds with a message that contains the status of the revocation from the CA.

Preconditions:

1. The certificate the EE wishes to revoke is not yet expired or revoked.

Message flow:

\[
\begin{array}{lll}
\text{Step#} & \text{EE} & (\text{L})\text{RA/CA} \\
1 & \text{format rr} & \rightarrow \text{rr} \rightarrow \\
2 & & \text{handle, re-protect or forward rr} \\
3 & & \text{receive rp} \\
4 & & \leftarrow \text{rp} \leftarrow \\
5 & \text{handle rp} & \\
\end{array}
\]

For this profile, the EE MUST include exactly one RevDetails structure in the rr. In case no error occurred the response to the rr MUST be an rp message. The (L)RA/CA MUST produce a rp containing a status field with a single set of values.

Detailed message description:

Revocation Request -- rr

Field                         Value

header
  -- As described in section 3.1

body
  -- The request of the EE to revoke its certificate
  rr                          REQUIRED
  -- MUST contain exactly one element of type RevDetails
  -- If more revocations are desired, further requests MUST be
  -- packaged in separate PKI Messages
  certDetails                 REQUIRED
  -- MUST be present and is of type CertTemplate
  serialNumber                REQUIRED
  -- MUST contain the certificate serialNumber attribute of the X.509
  -- certificate to be revoked
  issuer                      REQUIRED
  -- MUST contain the issuer attribute of the X.509 certificate to be
  -- revoked
  crlEntryDetails             REQUIRED
  -- MUST contain exactly one reasonCode of type CRLReason (see
  -- [RFC 5280] section 5.3.1)
  -- If the reason for this revocation is not known or shall not be
  -- published the reasonCode MUST be 0 = unspecified

protection                    REQUIRED
  -- As described in section 3.2 and the private key related to the
  -- certificate to be revoked

extraCerts                    REQUIRED
  -- As described in section 3.3

Revocation Response -- rp

Field                         Value

header
  -- As described in section 3.1

body
  -- The responds of the (L)RA/CA to the request as appropriate
rp REQUIRED
status REQUIRED
  -- MUST contain exactly one element of type PKIStatusInfo
status REQUIRED
  -- positive value allowed: "accepted"
-- negative value allowed: "rejection"
statusString OPTIONAL
  -- MAY be any human-readable text for debugging, logging or to
-- display in a GUI
failInfo OPTIONAL
  -- MAY be present if and only if status is "rejection"

protection REQUIRED
  -- As described in section 3.2

extraCerts REQUIRED

5.3. Error reporting

This functionality should be used by an EE to report any error
conditions upstream to the (L)RA/CA. Error reporting by the (L)RA
downstream to the EE is described in Section 6.3.

In case the error condition is related to specific details of an ip,
cp, or kup response message and a confirmation is expected the error
condition MUST be reported in the respective certConf message with
negative contents.

General error conditions, e.g., problems with the message header,
protection, or extraCerts, and negative feedback on rp, pollRep, or
pkiConf messages MAY be reported in the form of an error message.

In both situations the error is reported in the PKIStatusInfo
structure of the respective message.

The (L)RA/CA MUST respond to an error message with a pkiConf message,
or with another error message if any part of the header is not valid.
Both sides MUST treat this message as the end of the current
transaction.

The PKIStatusInfo structure is used to report errors. The
PKIStatusInfo structure SHOULD consist of the following fields:

  o status: Here the PKIStatus value rejection is the only one
    allowed.
statusString: Here any human-readable valid value for logging or to display in a GUI SHOULD be added.

failInfo: Here the PKIFailureInfo values MAY be used in the following way. For explanation of the reason behind a specific value, please refer to [RFC4210] Appendix F.

* transactionIdInUse: This is sent in case the received request contains a transaction ID that is already in use for another transaction. An EE receiving such error message SHOULD resend the request in a new transaction using a different transaction ID.

* systemUnavail or systemFailure: This is sent in case a back-end system is not available or currently not functioning correctly. An EE receiving such error message SHOULD resend the request in a new transaction after some time.

Detailed error message description:

Error Message -- error

Field                      Value

header
   -- As described in section 3.1

body
   -- The message sent by the EE or the (L)RA/CA to indicate an error that occurred
   error                    REQUIRED
   pKIStatusInfo            REQUIRED
   status                   REQUIRED
   -- MUST have the value "rejection"
   statusString            RECOMMENDED
   -- SHOULD be any human-readable text for debugging, logging
   -- or to display in a GUI
   failInfo                 OPTIONAL
   -- MAY be present

protection                  REQUIRED
   -- As described in section 3.2

extraCerts                 OPTIONAL
   -- As described in section 3.3
5.4. Support messages

The following support messages offer on demand in-band transport of content that may be relevant to the EE. The general request messages and general response messages are used for this purpose.

The general message and general response transport InfoTypeAndValue structures. In addition to those infoType values defined in CMP [RFC4210] further OIDs MAY be defined to define new certificate management transactions, or general-purpose messages as needed in a specific environment.

Possible content described here address:

- Update of Root CA certificates
- Parameters needed for a planned certificate request message <TBD>
- Request an enrollment voucher

< Details need to be defined later >

5.4.1. Root CA certificate update

This message sequence can be used by an EE to request an update of a Root CA Certificate by the EE. It utilizes the root CA key update announcement message as described in [RFC4210] Appendix E.4 as response to a respective general request message.

An EE requests a root CA certificate update from the (L)RA/CA by sending a general message with OID id-it-caKeyUpdateInfo. The (L)RA/CA responds with a general response with the same OID that either contains the update of the root CA certificate consisting of three certificates, or with no content in case no update is available. These three certificates are described in more detail in section 4.4.1, section 6.2, and Appendix E.3 of [RFC4210].

< Details need to be defined later >

5.4.2. Get enrollment voucher

This message sequence can be used by an EE to request an enrollment voucher containing the root certificate of a new PKI to establish trust in this PKI, e.g., in case no out-of-band transport is available. Such an enrollment voucher can be used in advance to an enrollment to this new environment. It may contain further information depending on the use case.
An EE requests an enrollment voucher from the (L)RA/CA by sending a general message. The (L)RA/CA responds with a general response with the same OID that contains the voucher.

< Details need to be defined later >

6. LRA and RA focused certificate management use cases

This chapter focuses on the communication of PKI backend components with each other. Depending on the network and PKI solution design, these will either be an LRA, RA or CA.

Typically, an (L)RA forwards messages from downstream, but it may also reply to them itself. Besides forwarding of received messages an (L)RA could also need to revoke certificates of EEs, report errors, or may need to manage its own certificates.

< In CMP Updates [brockhaus-lamps-cmp-updates] additional extended key usages like id-kp-cmpRA will be defined to indicate that a key pair is entitled to be used for signature-based protection of a CMP message by an (L)RA/CA. >

6.1. Forwarding of messages

Each CMP request message (i.e., ir, cr, p10cr, kur, pollReq, or certConf) or error message coming from an EE or the previous (downstream) PKI component MUST be sent to the next (upstream) PKI component. This PKI component MUST forward response messages to the next (downstream) PKI component or EE.

The (L)RA SHOULD verify the protection, the syntax, the required message fields, the message type, and if applicable the authorization and the proof-of-possession of the message. Additional checks or actions MAY be applied depending on the PKI solution requirements and concept. If one of these verification procedures fails, the (L)RA SHOULD respond with a negative response message and SHOULD not forward the message further upstream. General error conditions should be handled as described in Section 5.3 and Section 6.3.

An (L)RA SHOULD not change the received message if not necessary. The (L)RA SHOULD only update the message protection if it is technically necessary. Concrete PKI system specifications may define in more detail if and when to do so.

This is particularly relevant in the upstream communication of a request message.
Each hop in a chain of PKI components has one or more functionalities, e.g.:

- An (L)RA may need to verify the identities of EEs or base authorization decisions for certification request processing on specific knowledge of the local setup, e.g., by consulting an inventory or asset management system.

- An (L)RA may need to add fields to certificate request messages.

- An (L)RA may need to store data from a message in a database for later usage or documentation purposes.

- An (L)RA may provide traversal of a network boundary.

- An (L)RA may need to double-check if the messages transferred back and forth are properly protected and well formed.

- An RA can collect messages from different LRAs and forward them to the CA.

- An (L)RA may provide a proof that it has performed all required checks.

- An (L)RA may initiate a delayed enrollment due to offline upstream communication or registration officer interaction.

- An (L)RA may grant the request of an EE to omit sending a confirmation message.

Therefore, the decision if a message should be forwarded

- unchanged with the original protection,

- unchanged with a new protection, or

- changed with a new protection

depends on the PKI solution design and the associated security policy (CP/CPS [RFC3647]).

This section specifies the different options an (L)RA may implement and use.

An (L)RA MAY update the protection of a message

- if the (L)RA performs changes to the header or the body of the message,
o if the (L)RA needs to prove checks or validations performed on the message to one of the next (upstream) PKI components,

o if the (L)RA needs to protect the message using a key and certificate from a different PKI, or

o if the (L)RA needs to replace a MAC based-protection.

This is particularly relevant in the upstream communication of certificate request messages.

The message protection covers only the header and the body and not the extraCerts. The (L)RA MAY change the extraCerts in any of the following message adaptations, e.g., to sort or add needed or to delete needless certificates to support the next hop. This may be particularly helpful to extend upstream messages with additional certificates or to reduce the number of certificates in downstream messages when forwarding to constrained devices.

6.1.1. Not changing protection

This message adaptation can be used by any (L)RA to forward an original CMP message without changing the header, body or protection. In any of these cases the (L)RA acts more like a proxy, e.g., on a network boundary, implementing no specific RA-like security functionality to the PKI.

This message adaptation MUST be used for forwarding kur messages that must not be approved by the respective (L)RA.

6.1.2. Replacing protection

The following two message adaptations can be used by any (L)RA to forward a CMP message with or without changes, but providing its own protection using its CMP signer key providing approval of this message. In this case the (L)RA acts as an actual Registration Authority (RA), which implements important security functionality of the PKI.

Before replacing the existing protection by a new protection, the (L)RA MUST verify the protection provided by the EE or by the previous PKI component and approve its content including any own modifications. For certificate requests the (L)RA MUST verify in particular the included proof-of-possession self-signature of the certTemplate using the public key of the requested certificate and MUST check that the EE, as authenticated by the message protection, is authorized to request a certificate with the subject as specified in the certTemplate.
In case the received message has been protected by a CA or another (L)RA, the current (L)RA MUST verify its protection and approve its content including any own modifications. For certificate requests the (L)RA MUST check that the other (L)RA, as authenticated by the message protection, is authorized to issue or forward the request.

These message adaptations MUST NOT be applied to kur request messages as described in Section 5.1.2 since their original protection using the key and certificate to be updated needs to be preserved, unless the regCtrl OldCertId is used to clearly identify the certificate to be updated.

6.1.2.1. Keeping proof-of-possession

This message adaptation can be used by any (L)RA to forward a CMP message with or without modifying the message header or body while preserving any included proof-of-possession.

By replacing the existing using its own CMP signer key the (L)RA provides a proof of verifying and approving of the message as described above.

In case the (L)RA modifies the certTemplate of an ir or cr message, the message adaptation in Section 6.1.2.2 needs to be applied instead.

6.1.2.2. Breaking proof-of-possession

This message adaptation can be used by any (L)RA to forward an ir or cr message with modifications of the certTemplate i.e., modification, addition, or removal of fields. Such changes will break the proof-of-possession provided by the EE in the original message.

By replacing the existing or applying an initial protection using its own CMP signer key the (L)RA provides a proof of verifying and approving the new message as described above.

In addition to the above the (L)RA MUST verify in particular the proof-of-possession contained in the original message as described above. If these checks were successfully performed the (L)RA MUST change the popo to raVerified.
The popo field MUST contain the raVerified choice in the certReq structure of the modified message as follows:

```
popo
  raVerified          REQUIRED
  -- MUST have the value NULL and indicates that the (L)RA
  -- verified the popo of the original message.
```

6.1.3. Initiating delayed enrollment

This message adaptation can be used by an (L)RA to initiate delayed enrollment. In this case a (L)RA/CA MUST add the status waiting in the response message. The (L)RA/CA MUST then reply to the pollReq messages as described in Section 5.1.6.

6.1.4. Granting omitted confirmation

This section will be removed though the functionality was incorporated into the standard enrollment use case in section Section 5.1.1 due to discussion with Tomas Gustavsson.

6.2. Revoking certificates on behalf of another’s entities

This message sequence can be used by an (L)RA to revoke a certificate of any other entity. This revocation request message MUST be signed by the (L)RA using its own CMP signer key to prove to the PKI authorization to revoke the certificate on behalf of the EE.

The general message flow for this profile is the same as given in section Section 5.2.

Preconditions:

1. the certificate to be revoked MUST be known to the (L)RA
2. the (L)RA MUST have the authorization to revoke the certificates of other entities issued by the corresponding CA

The profile for this exchange is identical to that given in section Section 5.2, with the following changes:

1. it is not required that the certificate to be revoked is not yet expired or revoked
2. the (L)RA acts as EE for this message exchange
3. the rr messages MUST be signed using the CMP signer key of the (L)RA.
6.3. Error reporting

This functionality should be used by the (L)RA to report any error conditions downstream to the EE. Potential error reporting by the EE upstream to the (L)RA/CA is described in Section 5.3.

In case the error condition is related to specific details of an ir, cr, pl0cr, or kur request message it MUST be reported in the specific response message, i.e., an ip, cp, or kup with negative contents.

General error conditions, e.g., problems with the message header, protection, or extraCerts, and negative feedback on rr, pollReq, certConf, or error messages MUST be reported in the form of an error message.

In both situations the (L)RA reports the errors in the PKIStatusInfo structure of the respective message as described in Section 5.3.

An EE receiving any such negative feedback SHOULD log the error appropriately and MUST terminate the current transaction.

7. CMP message transport variants

The CMP messages are designed to be self-contained, such that in principle any transport can be used. HTTP SHOULD be used for online transport while file-based transport MAY be used in case offline transport is required. In case HTTP transport is not desired or possible, CMP messages MAY also be piggybacked on any other reliable transport protocol, e.g., CoAP [RFC7252].

Independently of the means of transport it could happen that messages are lost, or a communication partner does not respond. In order to prevent waiting indefinitely, each CMP client component SHOULD use a configurable per-request timeout, and each CMP server component SHOULD use a configurable per-response timeout in case a further message is to be expected from the client side. In this way a hanging transaction can be closed cleanly with an error and related resources (for instance, any cached extraCerts) can be freed.

7.1. HTTP transport

This transport mechanism can be used by an EE and (L)RA/CA to transfer CMP messages over HTTP. If HTTP transport is used the specifications as described in [RFC6712] MUST be followed.
7.2. HTTPS transport using certificates

This transport mechanism can be used by an EE and (L)RA/CA to further protect the HTTP transport as described in Section 7.1 using TLS 1.2 [RFC5246] or TLS 1.3 [RFC8446] as described in [RFC2818] with certificate-based authentication. Using this transport mechanism, the CMP transport via HTTPS MUST use TLS server authentication and SHOULD use TLS client authentication.

**EE:**
- The EE SHOULD use a TLS client certificate as far as available. If no dedicated TLS certificate is available the EE SHOULD use an already existing certificate identifying the EE (e.g., a manufacturer certificate).
- If no TLS certificate is available at the EE, server-only authenticated TLS SHOULD be used.
- The EE MUST validate the TLS server certificate of its communication partner.

**(L)RA:**
- Each (L)RA SHOULD use a TLS client certificate on its upstream (client) interface.
- Each (L)RA SHOULD use a TLS server certificate on its downstream (server) interface.
- Each (L)RA MUST validate the TLS certificate of its communication partner.

NOTE: The requirements for checking certificates given in [RFC5280], [RFC5246] and [RFC8446] MUST be followed for the TLS layer. OCSP or CRLs SHOULD be used for status checking of the TLS certificates of communication partners.

7.3. HTTPS transport using shared secrets

This transport mechanism can be used by an EE and (L)RA/CA to further protect the HTTP transport as described in Section 7.1 using TLS 1.2 [RFC5246] or TLS 1.3 [RFC8446] as described in [RFC2818] with mutual authentication based on shared secrets as described in [RFC5054].

**EE:**
- The EE MUST use the shared symmetric key for authentication.
(L)RA:

- The (L)RA MUST use the shared symmetric key for authentication.

### 7.4. File-based transport

For offline transfer file-based transport MAY be used. Offline transport is typically used between LRA and RA nodes.

Connection and error handling mechanisms like those specified for HTTP in [RFC6712] need to be implemented.

< Details need to be defined later >

### 7.5. CoAP transport

In constrained environments where no HTTP transport is desired or possible, CoAP [RFC7252] MAY be used instead. Connection and error handling mechanisms like those specified for HTTP in [RFC6712] may need to be implemented.

Such specification is out of scope of this document and would need to be specifies in a separate document.

### 7.6. Piggybacking on other reliable transport

For online transfer where no HTTP transport is desired or possible CMP messages MAY also be transported on some other reliable protocol. Connection and error handling mechanisms like those specified for HTTP in [RFC6712] need to be implemented.

Such specification is out of scope of this document and would need to be specifies in a separate document, e.g. in the scope of the respective transport protocol used.

### 8. IANA Considerations

<Add any IANA considerations>

### 9. Security Considerations

<Add any security considerations>

### 10. Acknowledgements

We would like to thank the various reviewers of this CMP profile.
11. References

11.1. Normative References


11.2. Informative References


[IEC62443-3-3] International Electrotechnical Commission, "IEC 62443 Part 3-3 - System security requirements and security levels", IEC 62443-3-3, August 2013, <Informative References>.


Appendix A. Additional Stuff

This becomes an Appendix.

Authors’ Addresses

Hendrik Brockhaus
Siemens AG
Otto-Hahn-Ring 6
Munich 81739
Germany

Email: hendrik.brockhaus@siemens.com
URI: http://www.siemens.com/

Steffen Fries
Siemens AG
Otto-Hahn-Ring 6
Munich 81739
Germany

Email: steffen.fries@siemens.com
URI: http://www.siemens.com/
David von Oheimb  
Siemens AG  
Otto-Hahn-Ring 6  
Munich 81739  
Germany  

Email: david.von.oheimb@siemens.com  
URI: http://www.siemens.com/
Use of the HSS/LMS Hash-based Signature Algorithm in the Cryptographic Message Syntax (CMS)
<draft-ietf-lamps-cms-hash-sig-09>

Abstract

This document specifies the conventions for using the Hierarchical Signature System (HSS) / Leighton-Micali Signature (LMS) hash-based signature algorithm with the Cryptographic Message Syntax (CMS). In addition, the algorithm identifier and public key syntax are provided. The HSS/LMS algorithm is one form of hash-based digital signature; it is described in RFC 8554.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html
Copyright and License Notice

Copyright (c) 2019 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 . . . . . . . . . . . . . . . . . . . . . . . . . 3
   1.1. ASN.1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   1.2. Terminology . . . . . . . . . . . . . . . . . . . . . . . . 3
   1.3. Motivation . . . . . . . . . . . . . . . . . . . . . . . . . 3
2. HSS/LMS Hash-based Signature Algorithm Overview . . . . . . . 4
   2.1. Hierarchical Signature System (HSS) . . . . . . . . . . . 4
   2.2. Leighton-Micali Signature (LMS) . . . . . . . . . . . . . 5
   2.3. Leighton-Micali One-time Signature Algorithm (LM-OTS) . . 6
3. Algorithm Identifiers and Parameters . . . . . . . . . . . . . . 7
4. HSS/LMS Public Key Identifier . . . . . . . . . . . . . . . . . 8
5. Signed-data Conventions . . . . . . . . . . . . . . . . . . . . 8
6. Security Considerations . . . . . . . . . . . . . . . . . . . . 9
7. IANA Considerations . . . . . . . . . . . . . . . . . . . . . . 10
8. References . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
   8.1. Normative References . . . . . . . . . . . . . . . . . . . 10
   8.2. Informative References . . . . . . . . . . . . . . . . . . 11
Appendix: ASN.1 Module . . . . . . . . . . . . . . . . . . . . . 13
Acknowledgements . . . . . . . . . . . . . . . . . . . . . . . . 14
Author’s Address . . . . . . . . . . . . . . . . . . . . . . . . . 14
1. Introduction

This document specifies the conventions for using the Hierarchical Signature System (HSS) / Leighton-Micali Signature (LMS) hash-based signature algorithm with the Cryptographic Message Syntax (CMS) [CMS] signed-data content type. The LMS system provides a one-time digital signature that is a variant of Merkle Tree Signatures (MTS). The HSS is built on top of the LMS system to efficiently scale for a larger numbers of signatures. The HSS/LMS algorithm is one form of hash-based digital signature, and it is described in [HASHSIG]. The HSS/LMS signature algorithm can only be used for a fixed number of signing operations. The number of signing operations depends upon the size of the tree. The HSS/LMS signature algorithm uses small public keys, and it has low computational cost; however, the signatures are quite large. The HSS/LMS private key can be very small when the signer is willing to perform additional computation at signing time; alternatively, the private key can consume additional memory and provide a faster signing time. The HSS/LMS signatures [HASHSIG] are currently defined to use exclusively SHA-256 [SHS].

1.1. ASN.1

CMS values are generated using ASN.1 [ASN1-B], using the Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER) [ASN1-E].

1.2. Terminology

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.

1.3. Motivation

There have been recent advances in cryptanalysis and advances in the development of quantum computers. Each of these advances pose a threat to widely deployed digital signature algorithms.

Recent advances in cryptoanalysis [BH2013] and progress in the development of quantum computers [NAS2019] pose a threat to widely deployed digital signature algorithms. As a result, there is a need to prepare for a day that cryptosystems such as RSA and DSA that depend on discrete logarithm and factoring cannot be depended upon.

If large-scale quantum computers are ever built, these computers will be able to break many of the public-key cryptosystems currently in
A post-quantum cryptosystem [PQC] is a system that is secure against quantum computers that have more than a trivial number of quantum bits (qu-bits). It is open to conjecture when it will be feasible to build such computers; however, RSA, DSA, ECDSA, and EdDSA are all vulnerable if large-scale quantum computers come to pass.

The HSS/LMS signature algorithm does not depend on the difficulty of discrete logarithm or factoring, as a result these algorithms are considered to be post-quantum secure. One use of post-quantum secure signatures is the protection of software updates, perhaps using the format described in [FWPROT], to enable deployment of software that implements new cryptosystems.

2. HSS/LMS Hash-based Signature Algorithm Overview

Merkle Tree Signatures (MTS) are a method for signing a large but fixed number of messages. An MTS system depends on a one-time signature method and a collision-resistant hash function.

This specification makes use of the hash-based algorithm specified in [HASHSIG], which is the Leighton and Micali adaptation [LM] of the original Lamport-Diffie-Winternitz-Merkle one-time signature system [M1979][M1987][M1989a][M1989b].

As implied by the name, the hash-based signature algorithm depends on a collision-resistant hash function. The hash-based signature algorithm specified in [HASHSIG] currently uses only the SHA-256 one-way hash function [SHS], but it also establishes an IANA registry [IANA-LMS] to permit the registration of additional one-way hash functions in the future.

2.1. Hierarchical Signature System (HSS)

The MTS system specified in [HASHSIG] uses a hierarchy of trees. The Hierarchical N-time Signature System (HSS) allows subordinate trees to be generated when needed by the signer. Otherwise, generation of the entire tree might take weeks or longer.

An HSS signature as specified in [HASHSIG] carries the number of signed public keys (Nspk), followed by that number of signed public keys, followed by the LMS signature as described in Section 2.2. The public key for the top-most LMS tree is the public key of the HSS system. The LMS private key in the parent tree signs the LMS public key in the child tree, and the LMS private key in the bottom-most tree signs the actual message. The signature over the public key and the signature over the actual message are LMS signatures as described in Section 2.2.
The elements of the HSS signature value for a stand-alone tree (a top tree with no children) can be summarized as:

\[ u32str(0) | | lms_signature /* signature of message */ \]

where, \( u32str() \) and || are used as defined in [HASHSIG].

The elements of the HSS signature value for a tree with \( N_{spk} \) signed public keys can be summarized as:

\[ u32str(N_{spk}) | | \]
\[ signed_{public_key}[0] | | \]
\[ signed_{public_key}[1] | | \]
\[ \ldots \]
\[ signed_{public_key}[N_{spk}-2] | | \]
\[ signed_{public_key}[N_{spk}-1] | | \]
\[ lms_signature /* signature of message */ \]

where, as defined in Section 3.3 of [HASHSIG], the \( signed_{public_key} \) structure contains the \( lms_signature \) over the public key followed by the public key itself. Note that \( N_{spk} \) is the number of levels in the hierarchy of trees minus 1.

2.2. Leighton-Micali Signature (LMS)

Each tree in the system specified in [HASHSIG] uses the Leighton-Micali Signature (LMS) system. LMS systems have two parameters. The first parameter is the height of the tree, \( h \), which is the number of levels in the tree minus one. The [HASHSIG] specification supports five values for this parameter: \( h=5; h=10; h=15; h=20; \) and \( h=25 \). Note that there are \( 2^h \) leaves in the tree. The second parameter is the number of bytes output by the hash function, \( m \), which is the amount of data associated with each node in the tree. The [HASHSIG] specification supports only the SHA-256 hash function [SHS], with \( m=32 \). As a result, the [HASHSIG] specification supports five tree sizes; they are identified as:

- LMS_SHA256_M32_H5;
- LMS_SHA256_M32_H10;
- LMS_SHA256_M32_H15;
- LMS_SHA256_M32_H20; and
- LMS_SHA256_M32_H25.

The [HASHSIG] specification establishes an IANA registry [IANA-LMS] to permit the registration of additional hash functions and additional tree sizes in the future.
As specified in [HASHSIG], the LMS public key consists of four elements: the lms_algorithm_type from the list above, the otstype to identify the LM-OTS type as discussed in Section 2.3, the private key identifier (I) as described in Section 5.3 of [HASHSIG], and the m-byte string associated with the root node of the tree (T[1]).

The LMS public key can be summarized as:

\[
\text{u32str(lms\_algorithm\_type)} \| u32str(otstype) \| I \| T[1]
\]

As specified in [HASHSIG], an LMS signature consists of four elements: the number of the leaf (q) associated with the LM-OTS signature, an LM-OTS signature as described in Section 2.3, a typecode indicating the particular LMS algorithm, and an array of values that is associated with the path through the tree from the leaf associated with the LM-OTS signature to the root. The array of values contains the siblings of the nodes on the path from the leaf to the root but does not contain the nodes on the path itself. The array for a tree with height h will have h values. The first value is the sibling of the leaf, the next value is the sibling of the parent of the leaf, and so on up the path to the root.

The four elements of the LMS signature value can be summarized as:

\[
u32str(q) \| ots\_signature \| u32str(type) \| path[0] \| path[1] \| \ldots \| path[h-1]
\]

2.3. Leighton-Micali One-time Signature Algorithm (LM-OTS)

Merkle Tree Signatures (MTS) depend on a one-time signature method, and [HASHSIG] specifies the use of the LM-OTS, which has five parameters:

\[n\] - The length in bytes of the hash function output. [HASHSIG] supports only SHA-256 [SHS], with n=32.

\[H\] - A preimage-resistant hash function that accepts byte strings of any length, and returns an n-byte string.

\[w\] - The width in bits of the Winternitz coefficients. [HASHSIG] supports four values for this parameter: w=1; w=2; w=4; and w=8.

\[p\] - The number of n-byte string elements that make up the LM-OTS signature.
ls - The number of bits that are left-shifted in the final step of the checksum function, which is defined in Section 4.4 of [HASHSIG].

The values of p and ls are dependent on the choices of the parameters n and w, as described in Appendix B of [HASHSIG].

The [HASHSIG] specification supports four LM-OTS variants:

LMOTS_SHA256_N32_W1;
LMOTS_SHA256_N32_W2;
LMOTS_SHA256_N32_W4; and
LMOTS_SHA256_N32_W8.

The [HASHSIG] specification establishes an IANA registry [IANA-LMS] to permit the registration of additional variants in the future.

Signing involves the generation of C, an n-byte random value.

The LM-OTS signature value can be summarized as the identifier of the LM-OTS variant, the random value, and a sequence of hash values (y[0] through y[p-1]) that correspond to the elements of the public key as described in Section 4.5 of [HASHSIG]:

\[ \text{u32str(otstype)} \ || \ C \ || \ y[0] \ || \ ... \ || \ y[p-1] \]

3. Algorithm Identifiers and Parameters

The algorithm identifier for an HSS/LMS hash-based signatures is:

\[ \text{id-alg-hss-lms-hashsig OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) alg(3) 17 } } \]

When this object identifier is used for an HSS/LMS signature, the AlgorithmIdentifier parameters field MUST be absent (that is, the parameters are not present; the parameters are not set to NULL).

The signature value is a large OCTET STRING. The signature format is designed for easy parsing. Each format includes a counter and type codes that indirectly providing all of the information that is needed to parse the value during signature validation.

The signature value identifies the hash function used in the HSS/LMS tree. In [HASHSIG] only the SHA-256 hash function [SHS] is supported, but it also establishes an IANA registry [IANA-LMS] to permit the registration of additional hash functions in the future.
4. HSS/LMS Public Key Identifier

The AlgorithmIdentifier for an HSS/LMS public key uses the id-alg-hss-lms-hashsig object identifier, and the parameters field MUST be absent.

When this AlgorithmIdentifier appears in the SubjectPublicKeyInfo field of an X.509 certificate [RFC5280], the certificate key usage extension MAY contain digitalSignature, nonRepudiation, keyCertSign, and cRLSign; however, it MUST NOT contain other values.

```plaintext
pk-HSS-LMS-HashSig PUBLIC-KEY ::= {
  IDENTIFIER id-alg-hss-lms-hashsig
  KEY HSS-LMS-HashSig-PublicKey
  PARAMS ARE absent
  CERT-KEY-USAGE
    { digitalSignature, nonRepudiation, keyCertSign, cRLSign } }
```

Note that the id-alg-hss-lms-hashsig algorithm identifier is also referred to as id-alg-mts-hashsig. This synonym is based on the terminology used in an early draft of the document that became [HASHSIG].

The public key value is an OCTET STRING. Like the signature format, it is designed for easy parsing. The value is the number of levels in the public key, L, followed by the LMS public key.

The HSS/LMS public key value can be summarized as:

```plaintext
u32str(L) || lms_public_key
```

Note that the public key for the top-most LMS tree is the public key of the HSS system. When L=1, the HSS system is a single tree.

5. Signed-data Conventions

As specified in [CMS], the digital signature is produced from the message digest and the signer’s private key. The signature is computed over different values depending on whether signed attributes are absent or present.

When signed attributes are absent, the HSS/LMS signature is computed over the content. When signed attributes are present, a hash is computed over the content using the same hash function that is used in the HSS/LMS tree, and then a message-digest attribute is constructed to contain the resulting hash value, and then the result
of DER encoding the set of signed attributes (which MUST include a
content-type attribute and a message-digest attribute, and then the
HSS/LMS signature is computed over the DER-encoded output. In
summary:

IF (signed attributes are absent)
THEN HSS_LMS_Sign(content)
ELSE message-digest attribute = Hash(content);
    HSS_LMS_Sign(DER(SignedAttributes))

When using [HASHSIG], the fields in the SignerInfo are used as
follows:

digestAlgorithm MUST contain the one-way hash function used to in
the HSS/LMS tree. In [HASHSIG], SHA-256 is the only supported
hash function, but other hash functions might be registered in
the future. For convenience, the AlgorithmIdentifier for
SHA-256 from [PKIXASN1] is repeated here:

    mda-sha256 DIGEST-ALGORITHM ::= {
        IDENTIFIER id-sha256
        PARAMS TYPE NULL ARE preferredAbsent }

    id-sha256 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
        country(16) us(840) organization(1) gov(101) csor(3)
        nistAlgorithms(4) hashalgs(2) 1 }

signatureAlgorithm MUST contain id-alg-hss-lms-hashsig, and the
algorithm parameters field MUST be absent.

signature contains the single HSS signature value resulting from
the signing operation as specified in [HASHSIG].

6. Security Considerations

Implementations MUST protect the private keys. Compromise of the
private keys may result in the ability to forge signatures. Along
with the private key, the implementation MUST keep track of which
leaf nodes in the tree have been used. Loss of integrity of this
tracking data can cause a one-time key to be used more than once. As
a result, when a private key and the tracking data are stored on non-
volatile media or stored in a virtual machine environment, care must
be taken to preserve confidentiality and integrity.

When generating an LMS key pair, an implementation MUST generate each
key pair independently of all other key pairs in the HSS tree.

An implementation MUST ensure that a LM-OTS private key is used to
generate a signature only one time, and ensure that it cannot be used for any other purpose.

The generation of private keys relies on random numbers. The use of inadequate pseudo-random number generators (PRNGs) to generate these values can result in little or no security. An attacker may find it much easier to reproduce the PRNG environment that produced the keys, searching the resulting small set of possibilities, rather than brute force searching the whole key space. The generation of quality random numbers is difficult, and [RFC4086] offers important guidance in this area.

The generation of hash-based signatures also depends on random numbers. While the consequences of an inadequate pseudo-random number generator (PRNGs) to generate these values is much less severe than the generation of private keys, the guidance in [RFC4086] remains important.

When computing signatures, the same hash function SHOULD be used to compute the message digest of the content and the signed attributes, if they are present.

7.  IANA Considerations

SMI Security for S/MIME Module Identifier (1.2.840.113549.1.9.16.0) registry, change the reference for value 64 to point to this document.

In the SMI Security for S/MIME Algorithms (1.2.840.113549.1.9.16.3) registry, change the description for value 17 to "id-alg-hss-lms-hashsig" and change the reference to point to this document.

Also, add the following note to the registry:

Value 17, "id-alg-hss-lms-hashsig", is also referred to as "id-alg-mts-hashsig".

8.  References

8.1.  Normative References

8.2. Informative References


Appendix: ASN.1 Module

```asn1
<CODE STARTS>

MTS-HashSig-2013
{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  id-smime(16) id-mod(0) id-mod-mts-hashsig-2013(64) }

DEFINITIONS IMPLICIT TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS
PUBLIC-KEY, SIGNATURE-ALGORITHM, SMIME-CAPS
FROM AlgorithmInformation-2009 -- RFC 5911 [CMSASN1]
{ iso(1) identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) id-mod(0)
  id-mod-algorithmInformation-02(58) } ;

--
-- Object Identifiers
--

id-alg-hss-lms-hashsig OBJECT IDENTIFIER ::= { iso(1)
  member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  smime(16) alg(3) 17 }

id-alg-mts-hashsig OBJECT IDENTIFIER ::= id-alg-hss-lms-hashsig

--
-- Signature Algorithm and Public Key
--

sa-HSS-LMS-HashSig SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-alg-hss-lms-hashsig
  PARAMS ARE absent
  PUBLIC-KEYS { pk-HSS-LMS-HashSig }
  SMIME-CAPS { IDENTIFIED BY id-alg-hss-lms-hashsig } }

<CODE ENDS>
```

Housley
pk-HSS-LMS-HashSig PUBLIC-KEY ::= {
   IDENTIFIER id-alg-hss-lms-hashsig
   KEY HSS-LMS-HashSig-PublicKey
   PARAMS ARE absent
   CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
}

HSS-LMS-HashSig-PublicKey ::= OCTET STRING

--
-- Expand the signature algorithm set used by CMS [CMSASN1U]
--

SignatureAlgorithmSet SIGNATURE-ALGORITHM ::= { sa-HSS-LMS-HashSig, ... }

--
-- Expand the S/MIME capabilities set used by CMS [CMSASN1]
--

SMimeCaps SMIME-CAPS ::= { sa-HSS-LMS-HashSig.&smimeCaps, ... }

END

<CODE ENDS>

Acknowledgements

Many thanks to Scott Fluhrer, Jonathan Hammell, Panos Kampanakis, John Mattsson, Jim Schaad, Sean Turner, Daniel Van Geest, Roman Danyliw, Dale Worley, and Joe Clarke for their careful review and comments.

Author’s Address

Russ Housley
Vigil Security, LLC
516 Dranesville Road
Herndon, VA 20170
USA

EMail: housley@vigilsec.com
Abstract

The invention of a large-scale quantum computer would pose a serious challenge for the cryptographic algorithms that are widely deployed today. The Cryptographic Message Syntax (CMS) supports key transport and key agreement algorithms that could be broken by the invention of such a quantum computer. By storing communications that are protected with the CMS today, someone could decrypt them in the future when a large-scale quantum computer becomes available. Once quantum-secure key management algorithms are available, the CMS will be extended to support the new algorithms, if the existing syntax does not accommodate them. In the near-term, this document describes a mechanism to protect today’s communication from the future invention of a large-scale quantum computer by mixing the output of key transport and key agreement algorithms with a pre-shared key.

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) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
1.  Introduction

The invention of a large-scale quantum computer would pose a serious challenge for the cryptographic algorithms that are widely deployed today [S1994]. It is an open question whether or not it is feasible to build a large-scale quantum computer, and if so, when that might happen [NAS2019]. However, if such a quantum computer is invented, many of the cryptographic algorithms and the security protocols that use them would become vulnerable.
The Cryptographic Message Syntax (CMS) [RFC5652][RFC5083] supports key transport and key agreement algorithms that could be broken by the invention of a large-scale quantum computer [C2PQ]. These algorithms include RSA [RFC8017], Diffie-Hellman [RFC2631], and Elliptic Curve Diffie-Hellman [RFC5753]. As a result, an adversary that stores CMS-protected communications today, could decrypt those communications in the future when a large-scale quantum computer becomes available.

Once quantum-secure key management algorithms are available, the CMS will be extended to support them, if the existing syntax does not already accommodate the new algorithms.

In the near-term, this document describes a mechanism to protect today’s communication from the future invention of a large-scale quantum computer by mixing the output of existing key transport and key agreement algorithms with a pre-shared key (PSK). Secure communication can be achieved today by mixing a strong PSK with the output of an existing key transport algorithm, like RSA [RFC8017], or an existing key agreement algorithm, like Diffie-Hellman [RFC2631] or Elliptic Curve Diffie-Hellman [RFC5753]. A security solution that is believed to be quantum resistant can be achieved by using a PSK with sufficient entropy along with a quantum resistant key derivation function (KDF), like HKDF [RFC5869], and a quantum resistant encryption algorithm, like 256-bit AES [AES]. In this way, today’s CMS-protected communication can be resistant to an attacker with a large-scale quantum computer.

In addition, there may be other reasons for including a strong PSK besides protection against the future invention of a large-scale quantum computer. For example, there is always the possibility of a cryptoanalytic breakthrough on one or more of the classic public-key algorithm, and there are longstanding concerns about undisclosed trapdoors in Diffie-Hellman parameters [FGHT2016]. Inclusion of a strong PSK as part of the overall key management offer additional protection against these concerns.

Note that the CMS also supports key management techniques based on symmetric key-encryption keys and passwords, but they are not discussed in this document because they are already quantum resistant. The symmetric key-encryption key technique is quantum resistant when used with an adequate key size. The password technique is quantum resistant when used with a quantum-resistant key derivation function and a sufficiently large password.
1.1. Terminology

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.

1.2. ASN.1

CMS values are generated using ASN.1 [X680], which uses the Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER) [X690].

1.3. Version Numbers

The major data structures include a version number as the first item in the data structure. The version number is intended to avoid ASN.1 decode errors. Some implementations do not check the version number prior to attempting a decode, and then if a decode error occurs, the version number is checked as part of the error handling routine. This is a reasonable approach; it places error processing outside of the fast path. This approach is also forgiving when an incorrect version number is used by the sender.

Whenever the structure is updated, a higher version number will be assigned. However, to ensure maximum interoperability, the higher version number is only used when the new syntax feature is employed. That is, the lowest version number that supports the generated syntax is used.

2. Overview

The CMS enveloped-data content type [RFC5652] and the CMS authenticated-enveloped-data content type [RFC5083] support both key transport and key agreement public-key algorithms to establish the key used to encrypt the content. No restrictions are imposed on the key transport or key agreement public-key algorithms, which means that any key transport or key agreement algorithm can be used, including algorithms that are specified in the future. In both cases, the sender randomly generates the content-encryption key, and then all recipients obtain that key. All recipients use the sender-generated symmetric content-encryption key for decryption.

This specification defines two quantum-resistant ways to establish a symmetric key-encryption key, which is used to encrypt the sender-generated content-encryption key. In both cases, the PSK is used as one of the inputs to a key-derivation function to create a quantum-
resistant key-encryption key. The PSK MUST be distributed to the sender and all of the recipients by some out-of-band means that does not make it vulnerable to the future invention of a large-scale quantum computer, and an identifier MUST be assigned to the PSK. It is best if each PSK has a unique identifier; however, if a recipient has more than one PSK with the same identifier, the recipient can try each of them in turn. A PSK is expected to be used with many messages, with a lifetime of weeks or months.

The content-encryption key or content-authenticated-encryption key is quantum-resistant, and the sender establishes it using these steps:

When using a key transport algorithm:

1. The content-encryption key or the content-authenticated-encryption key, called CEK, is generated at random.

2. The key-derivation key, called KDK, is generated at random.

3. For each recipient, the KDK is encrypted in the recipient’s public key, then the key derivation function (KDF) is used to mix the pre-shared key (PSK) and the KDK to produce the key-encryption key, called KEK.

4. The KEK is used to encrypt the CEK.

When using a key agreement algorithm:

1. The content-encryption key or the content-authenticated-encryption key, called CEK, is generated at random.

2. For each recipient, a pairwise key-encryption key, called KEK1, is established using the recipient’s public key and the sender’s private key. Note that KEK1 will be used as a key-derivation key.

3. For each recipient, the key derivation function (KDF) is used to mix the pre-shared key (PSK) and the pairwise KEK1, and the result is called KEK2.

4. For each recipient, the pairwise KEK2 is used to encrypt the CEK.

As specified in Section 6.2.5 of [RFC5652], recipient information for additional key management techniques are represented in the OtherRecipientInfo type. Two key management techniques are specified in this document, and they are each identified by a unique ASN.1 object identifier.
The first key management technique, called keyTransPSK, see Section 3, uses a key transport algorithm to transfer the key-derivation key from the sender to the recipient, and then the key-derivation key is mixed with the PSK using a KDF. The output of the KDF is the key-encryption key, which is used for the encryption of the content-encryption key or content-authenticated-encryption key.

The second key management technique, called keyAgreePSK, see Section 4, uses a key agreement algorithm to establish a pairwise key-encryption key, which is then mixed with the PSK using a KDF to produce a second pairwise key-encryption key, which is then used to encrypt the content-encryption key or content-authenticated-encryption key.

3. keyTransPSK

Per-recipient information using keyTransPSK is represented in the KeyTransPSKRecipientInfo type, which is indicated by the id-ori-keyTransPSK object identifier. Each instance of KeyTransPSKRecipientInfo establishes the content-encryption key or content-authenticated-encryption key for one or more recipients that have access to the same PSK.

The id-ori-keyTransPSK object identifier is:

   id-ori OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) TBD1 }

   id-ori-keyTransPSK OBJECT IDENTIFIER ::= { id-ori 1 }

The KeyTransPSKRecipientInfo type is:

KeyTransPSKRecipientInfo ::= SEQUENCE {
   version CMSVersion, -- always set to 0
   pskid PreSharedKeyIdentifier,
   kdfAlgorithm KeyDerivationAlgorithmIdentifier,
   keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
   ktris KeyTransRecipientInfos,
   encryptedKey EncryptedKey
}

PreSharedKeyIdentifier ::= OCTET STRING

KeyTransRecipientInfos ::= SEQUENCE OF KeyTransRecipientInfo
The fields of the KeyTransPSKRecipientInfo type have the following meanings:

version is the syntax version number. The version MUST be 0. The CMSVersion type is described in Section 10.2.5 of [RFC5652].

pskid is the identifier of the PSK used by the sender. The identifier is an OCTET STRING, and it need not be human readable.

kdfAlgorithm identifies the key-derivation algorithm, and any associated parameters, used by the sender to mix the key-derivation key and the PSK to generate the key-encryption key. The KeyDerivationAlgorithmIdentifier is described in Section 10.1.6 of [RFC5652].

keyEncryptionAlgorithm identifies a key-encryption algorithm used to encrypt the content-encryption key. The KeyEncryptionAlgorithmIdentifier is described in Section 10.1.3 of [RFC5652].

ktris contains one KeyTransRecipientInfo type for each recipient; it uses a key transport algorithm to establish the key-derivation key. That is, the encryptedKey field of KeyTransRecipientInfo contains the key-derivation key instead of the content-encryption key. KeyTransRecipientInfo is described in Section 6.2.1 of [RFC5652].

encryptedKey is the result of encrypting the content-encryption key or the content-authenticated-encryption key with the key-encryption key. EncryptedKey is an OCTET STRING.

4. keyAgreePSK

Per-recipient information using keyAgreePSK is represented in the KeyAgreePSKRecipientInfo type, which is indicated by the id-ori-keyAgreePSK object identifier. Each instance of KeyAgreePSKRecipientInfo establishes the content-encryption key or content-authenticated-encryption key for one or more recipients that have access to the same PSK.

The id-ori-keyAgreePSK object identifier is:


id-ori-keyAgreePSK OBJECT IDENTIFIER ::= { id-ori 2 }
The KeyAgreePSKRecipientInfo type is:

KeyAgreePSKRecipientInfo ::= SEQUENCE {
    version CMSVersion,  -- always set to 0
    pskid PreSharedKeyIdentifier,
    originator [0] EXPLICIT OriginatorIdentifierOrKey,
    ukm [1] EXPLICIT UserKeyingMaterial OPTIONAL,
    kdfAlgorithm KeyDerivationAlgorithmIdentifier,
    keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
    recipientEncryptedKeys RecipientEncryptedKeys }

The fields of the KeyAgreePSKRecipientInfo type have the following meanings:

version is the syntax version number. The version MUST be 0. The CMSVersion type is described in Section 10.2.5 of [RFC5652].

pskid is the identifier of the PSK used by the sender. The identifier is an OCTET STRING, and it need not be human readable.

originator is a CHOICE with three alternatives specifying the sender’s key agreement public key. Implementations MUST support all three alternatives for specifying the sender’s public key. The sender uses their own private key and the recipient’s public key to generate a pairwise key-encryption key. A key derivation function (KDF) is used to mix the PSK and the pairwise key-encryption key to produce a second key-encryption key. The OriginatorIdentifierOrKey type is described in Section 6.2.2 of [RFC5652].

ukm is optional. With some key agreement algorithms, the sender provides a User Keying Material (UKM) to ensure that a different key is generated each time the same two parties generate a pairwise key. Implementations MUST accept a KeyAgreePSKRecipientInfo SEQUENCE that includes a ukm field. Implementations that do not support key agreement algorithms that make use of UKMs MUST gracefully handle the presence of UKMs. The UserKeyingMaterial type is described in Section 10.2.6 of [RFC5652].

kdfAlgorithm identifies the key-derivation algorithm, and any associated parameters, used by the sender to mix the pairwise key-encryption key and the PSK to produce a second key-encryption key of the same length as the first one. The KeyDerivationAlgorithmIdentifier is described in Section 10.1.6 of [RFC5652].
keyEncryptionAlgorithm identifies a key-encryption algorithm used to encrypt the content-encryption key or the content-authenticated-encryption key. The KeyEncryptionAlgorithmIdentifier type is described in Section 10.1.3 of [RFC5652].

recipientEncryptedKeys includes a recipient identifier and encrypted key for one or more recipients. The KeyAgreeRecipientIdentifier is a CHOICE with two alternatives specifying the recipient’s certificate, and thereby the recipient’s public key, that was used by the sender to generate a pairwise key-encryption key. The encryptedKey is the result of encrypting the content-encryption key or the content-authenticated-encryption key with the second pairwise key-encryption key. EncryptedKey is an OCTET STRING. The RecipientEncryptedKeys type is defined in Section 6.2.2 of [RFC5652].

5. Key Derivation

Many key derivation functions (KDFs) internally employ a one-way hash function. When this is the case, the hash function that is used is indirectly indicated by the KeyDerivationAlgorithmIdentifier. HKDF [RFC5869] is one example of a KDF that makes use of a hash function.

Other KDFs internally employ an encryption algorithm. When this is the case, the encryption that is used is indirectly indicated by the KeyDerivationAlgorithmIdentifier. For example, AES-128-CMAC can be used for randomness extraction in a KDF as described in [NIST2018].

A KDF has several input values. This section describes the conventions for using the KDF to compute the key-encryption key for KeyTransPSKRecipientInfo and KeyAgreePSKRecipientInfo. For simplicity, the terminology used in the HKDF [RFC5869] specification is used here.

The KDF inputs are:

IKM is the input keying material; it is the symmetric secret input to the KDF. For KeyTransPSKRecipientInfo, it is the key-derivation key. For KeyAgreePSKRecipientInfo, it is the pairwise key-encryption key produced by the key agreement algorithm.

salt is an optional non-secret random value. Many KDFs do not require a salt, and the KeyDerivationAlgorithmIdentifier assignments for HKDF [RFC8619] do not offer a parameter for a salt. If a particular KDF requires a salt, then the salt value is provided as a parameter of the KeyDerivationAlgorithmIdentifier.
L is the length of output keying material in octets; the value depends on the key-encryption algorithm that will be used. The algorithm is identified by the KeyEncryptionAlgorithmIdentifier. In addition, the OCTET IDENTIFIER portion of the KeyEncryptionAlgorithmIdentifier is included in the next input value, called info.

info is optional context and application specific information. The DER-encoding of CMSORIforPSKOtherInfo is used as the info value, and the PSK is included in this structure. Note that EXPLICIT tagging is used in the ASN.1 module that defines this structure. For KeyTransPSKRecipientInfo, the ENUMERATED value of 5 is used. For KeyAgreePSKRecipientInfo, the ENUMERATED value of 10 is used. CMSORIforPSKOtherInfo is defined by the following ASN.1 structure:

\[
\text{CMSORIforPSKOtherInfo ::= SEQUENCE }\
\text{ { psk OCTET STRING, } } \\
\text{ keyMgmtAlgType ENUMERATED { } } \\
\text{ keyTrans } \\
\text{ (5), } \\
\text{ keyAgree } \\
\text{ (10) }, \\
\text{ keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier, } \\
\text{ pskLength INTEGER (1..MAX), } \\
\text{ kdkLength INTEGER (1..MAX) ) } \\
\]

The fields of type CMSORIforPSKOtherInfo have the following meanings:

psk is an OCTET STRING; it contains the PSK.

keyMgmtAlgType is either set to 5 or 10. For KeyTransPSKRecipientInfo, the ENUMERATED value of 5 is used. For KeyAgreePSKRecipientInfo, the ENUMERATED value of 10 is used.

keyEncryptionAlgorithm is the KeyEncryptionAlgorithmIdentifier, which identifies the algorithm and provides algorithm parameters, if any.

pskLength is a positive integer; it contains the length of the PSK in octets.

kdkLength is a positive integer; it contains the length of the key-derivation key in octets. For KeyTransPSKRecipientInfo, the key-derivation key is generated by the sender. For KeyAgreePSKRecipientInfo, the key-derivation key is the pairwise key-encryption key produced by the key agreement algorithm.
The KDF output is:

OKM is the output keying material, which is exactly L octets. The
OKM is the key-encryption key that is used to encrypt the content-
encryption key or the content-authenticated-encryption key.

An acceptable KDF MUST accept IKM, L, and info inputs; and acceptable
KDF MAY also accept salt and other inputs. All of these inputs MUST
influence the output of the KDF. If the KDF requires a salt or other
inputs, then those inputs MUST be provided as parameters of the
KeyDerivationAlgorithmIdentifier.

6. ASN.1 Module

This section contains the ASN.1 module for the two key management
techniques defined in this document. This module imports types from
other ASN.1 modules that are defined in [RFC5912] and [RFC6268].

<CODE BEGINS>

CMSORIforPSK-2019
{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
  smime(16) modules(0) id-mod-cms-ori-psk-2019(TBD0) }
DEFINITIONS EXPLICIT TAGS :=
BEGIN
  -- EXPORTS All
IMPORTS
  AlgorithmIdentifier(), KEY-DERIVATION
  FROM AlgorithmInformation-2009 -- [RFC5912]
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-algorithmInformation-02(58) }
  OTHER-RECIPIENT, OtherRecipientInfo, CMSVersion,
  KeyTransRecipientInfo, OriginatorIdentifierOrKey,
  UserKeyingMaterial, RecipientEncryptedKeys, EncryptedKey,
  KeyDerivationAlgorithmIdentifier, KeyEncryptionAlgorithmIdentifier
  FROM CryptographicMessageSyntax-2010 -- [RFC6268]
  { iso(1) member-body(2) us(840) rsadsi(113549)
    pkcs(1) pkcs-9(9) smime(16) modules(0)
    id-mod-cms-2009(58) };

Housley                                                        [Page 11]
SupportedOtherRecipInfo OTHER-RECIPIENT ::= {
    ori-keyTransPSK,
    ori-keyAgreePSK,
    ...
}

-- Key Transport with Pre-Shared Key
--
ori-keyTransPSK OTHER-RECIPIENT ::= {
    KeyTransPSKRecipientInfo IDENTIFIED BY id-ori-keyTransPSK
}

id-ori OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
    rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) TBD1 }

id-ori-keyTransPSK OBJECT IDENTIFIER ::= { id-ori 1 }

KeyTransPSKRecipientInfo ::= SEQUENCE {
    version CMSVersion,  -- always set to 0
    pskid PreSharedKeyIdentifier,
    kdfAlgorithm KeyDerivationAlgorithmIdentifier,
    keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
    ktris KeyTransRecipientInfos,
    encryptedKey EncryptedKey
}

PreSharedKeyIdentifier ::= OCTET STRING

KeyTransRecipientInfos ::= SEQUENCE OF KeyTransRecipientInfo

-- Key Agreement with Pre-Shared Key
--
ori-keyAgreePSK OTHER-RECIPIENT ::= {
    KeyAgreePSKRecipientInfo IDENTIFIED BY id-ori-keyAgreePSK
}

id-ori-keyAgreePSK OBJECT IDENTIFIER ::= { id-ori 2 }
KeyAgreePSKRecipientInfo ::= SEQUENCE {
    version CMSVersion, -- always set to 0
    pskid PreSharedKeyIdentifier,
    originator [0] EXPLICIT OriginatorIdentifierOrKey,
    ukm [1] EXPLICIT UserKeyingMaterial OPTIONAL,
    kdfAlgorithm KeyDerivationAlgorithmIdentifier,
    keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
    recipientEncryptedKeys RecipientEncryptedKeys }

--
-- Structure to provide 'info' input to the KDF,
-- including the Pre-Shared Key
--

CMSORIforPSKOtherInfo ::= SEQUENCE {
    psk OCTET STRING,
    keyMgmtAlgType ENUMERATED {
        keyTrans (5),
        keyAgree (10) },
    keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
    pskLength INTEGER (1..MAX),
    kdkLength INTEGER (1..MAX) }

END

<CODE ENDS>

7. Security Considerations

The security considerations in related to the CMS enveloped-data content type in [RFC5652] and the security considerations related to the CMS authenticated-enveloped-data content type in [RFC5083] continue to apply.

Implementations of the key derivation function must compute the entire result, which in this specification is a key-encryption key, before outputting any portion of the result. The resulting key-encryption key must be protected. Compromise of the key-encryption key may result in the disclosure of all content-encryption keys or content-authenticated-encryption keys that were protected with that keying material, which in turn may result in the disclosure of the content. Note that there are two key-encryption keys when a PSK with a key agreement algorithm is used, with similar consequence for the compromise of either one of these keys.

Implementations must protect the pre-shared key (PSK), key transport private key, the agreement private key, and the key-derivation key. Compromise of the PSK will make the encrypted content vulnerable to
the future invention of a large-scale quantum computer. Compromise of the PSK and either the key transport private key or the agreement private key may result in the disclosure of all contents protected with that combination of keying material. Compromise of the PSK and the key-derivation key may result in disclosure of all contents protected with that combination of keying material.

A large-scale quantum computer will essentially negate the security provided by the key transport algorithm or the key agreement algorithm, which means that the attacker with a large-scale quantum computer can discover the key-derivation key. In addition a large-scale quantum computer effectively cuts the security provided by a symmetric key algorithm in half. Therefore, the PSK needs at least 256 bits of entropy to provide 128 bits of security. To match that same level of security, the key derivation function needs to be quantum-resistant and produce a key-encryption key that is at least 256 bits in length. Similarly, the content-encryption key or content-authenticated-encryption key needs to be at least 256 bits in length.

When using a PSK with a key transport or a key agreement algorithm, a key-encryption key is produced to encrypt the content-encryption key or content-authenticated-encryption key. If the key-encryption algorithm is different than the algorithm used to protect the content, then the effective security is determined by the weaker of the two algorithms. If, for example, content is encrypted with 256-bit AES, and the key is wrapped with 128-bit AES, then at most 128 bits of protection is provided. Implementers must ensure that the key-encryption algorithm is as strong or stronger than the content-encryption algorithm or content-authenticated-encryption algorithm.

The selection of the key-derivation function imposes an upper bound on the strength of the resulting key-encryption key. The strength of the selected key-derivation function should be at least as strong as the key-encryption algorithm that is selected. NIST SP 800-56C Revision 1 [NIST2018] offers advice on the security strength of several popular key-derivation functions.

Implementers should not mix quantum-resistant key management algorithms with their non-quantum-resistant counterparts. For example, the same content should not be protected with KeyTransRecipientInfo and KeyTransPSKRecipientInfo. Likewise, the same content should not be protected with KeyAgreeRecipientInfo and KeyAgreePSKRecipientInfo. Doing so would make the content vulnerable to the future invention of a large-scale quantum computer.

Implementers should not send the same content in different messages,
one using a quantum-resistant key management algorithm and the other
using a non-quantum-resistant key management algorithm, even if the
content-encryption key is generated independently. Doing so may
allow an eavesdropper to correlate the messages, making the content
vulnerable to the future invention of a large-scale quantum computer.

This specification does not require that PSK is known only by the
sender and recipients. The PSK may be known to a group. Since
confidentiality depends on the key transport or key agreement
algorithm, knowledge of the PSK by other parties does not enable
inherently eavesdropping. However, group members can record the
traffic of other members, and then decrypt it if they ever gain
access to a large-scale quantum computer. Also, when many parties
know the PSK, there are many opportunities for theft of the PSK by an
attacker. Once an attacker has the PSK, they can decrypt stored
traffic if they ever gain access to a large-scale quantum computer in
the same manner as a legitimate group member.

Sound cryptographic key hygiene is to use a key for one and only one
purpose. Use of the recipient’s public key for both the traditional
CMS and the PSK-mixing variation specified in this document would be
a violation of this principle; however, there is no known way for an
attacker to take advantage of this situation. That said, an
application should enforce separation whenever possible. For
example, a purpose identifier for use in the X.509 extended key usage
certificate extension [RFC5280] could be identified in the future to
indicate that a public key should only be used in conjunction with a
PSK, or only without.

Implementations must randomly generate key-derivation keys as well as
the content-encryption keys or content-authenticated-encryption keys.
Also, the generation of public/private key pairs for the key
transport and key agreement algorithms rely on a random numbers. The
use of inadequate pseudo-random number generators (PRNGs) to generate
cryptographic keys can result in little or no security. An attacker
may find it much easier to reproduce the PRNG environment that
produced the keys, searching the resulting small set of
possibilities, rather than brute force searching the whole key space.
The generation of quality random numbers is difficult. [RFC4086]
offers important guidance in this area.

Implementers should be aware that cryptographic algorithms become
weaker with time. As new cryptanalysis techniques are developed and
computing performance improves, the work factor to break a particular
cryptographic algorithm will be reduced. Therefore, cryptographic
algorithm implementations should be modular, allowing new algorithms
to be readily inserted. That is, implementers should be prepared for
the set of supported algorithms to change over time.
The security properties provided by the mechanisms specified in this
document can be validated using formal methods. A ProVerif proof in
[H2019] shows that an attacker with a large-scale quantum computer
that is capable of breaking the Diffie-Hellman key agreement
algorithm cannot disrupt the delivery of the content-encryption key
to the recipient and the attacker cannot learn the content-encryption
key from the protocol exchange.

8. Privacy Considerations

An observer can see which parties are using each PSK simply by
watching the PSK key identifiers. However, the addition of these key
identifiers is not really making privacy worse. When key transport
is used, the RecipientIdentifier is always present, and it clearly
identifies each recipient to an observer. When key agreement is
used, either the IssuerAndSerialNumber or the RecipientKeyIdentifier
is always present, and these clearly identify each recipient.

9. IANA Considerations

One object identifier for the ASN.1 module in Section 6 was assigned
in the SMI Security for S/MIME Module Identifiers
(1.2.840.113549.1.9.16.0) [IANA-MOD] registry:

```
id-mod-cms-ori-psk-2019 OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
    pkcs-9(9) smime(16) mod(0) TBD0 }
```

One new registry was created for Other Recipient Info Identifiers
within the SMI Security for S/MIME Mail Security
(1.2.840.113549.1.9.16) [IANA-SMIME] registry:

```
id-ori OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
    rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) TBD1 }
```

Updates to the new registry are to be made according to the
Specification Required policy as defined in [RFC8126]. The expert is
expected to ensure that any new values identify additions
RecipientInfo structures for use with the CMS. Object identifiers
for other purposes should not be assigned in this arc.
Two assignments were made in the new SMI Security for Other Recipient Info Identifiers (1.2.840.113549.1.9.16.TBD1) [IANA-ORI] registry with references to this document:

```
   id-ori-keyTransPSK OBJECT IDENTIFIER ::= {
      iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
      pkcs-9(9) smime(16) id-ori(TBD1) 1 }

   id-ori-keyAgreePSK OBJECT IDENTIFIER ::= {
      iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
      pkcs-9(9) smime(16) id-ori(TBD1) 2 }
```

10. References

10.1. Normative References


[RFC5083] Housley, R., "Cryptographic Message Syntax (CMS)
Authenticated-Enveloped-Data Content Type", RFC 5083, November 2007.


10.2. Informative References


[IANA-MOD] https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#security-smime-0.


Appendix A: Key Transport with PSK Example

This example shows the establishment of an AES-256 content-encryption key using:
- a pre-shared key of 256 bits;
- key transport using RSA PKCS#1 v1.5 with a 3072-bit key;
- key derivation using HKDF with SHA-384; and
- key wrap using AES-256-KEYWRAP.

In real-world use, the originator would encrypt the key-derivation key in their own RSA public key as well as the recipient’s public key. This is omitted in an attempt to simplify the example.

A.1. Originator Processing Example

The pre-shared key known to Alice and Bob, in hexadecimal:

c244cdd11a0d1f39d9b61282770244fb0f6befb91ab7f96cb05213365cf95b15
The identifier assigned to the pre-shared key is:
ptf-kmc:13614122112

Alice obtains Bob’s public key:

-----BEGIN PUBLIC KEY-----
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEA3ocW14cxncPJ47fnEjBZ
AyfC21qapLSET4jvV6C7gGeVzRqXWDwl+cfYBBR2ej3j3/0ecDmu+XuVi2+s5JH
Keexa+ifuhzs3yifgeEpeK8+5usHhn20/NBLhYkhb3k1AcCgQ56dpDrDvDcLqQ
vS5jgj/VO/Opn2bofHC0eovt8Q/roahrJe1Pl1yQ4udWB8zEz<t>ezj4mLlfobA9y1YaYx
2AHHZJeo3mnRn1gJXo6mE00E/6qkhJDHSMDm12WG6m09TCDZc9y13cAJDU6Ir0v
SH7qU8/vN13y4OOFkn8hM4kmZ6bJqZt5NhjHtY4uQ0VMW3RyEShrO02mrp39a
ulNhH3EXdxA1t75H3qC7zJaSwEWMJyqOE3yFEGRKn8fxubj1716D8UecAxAzFy
FL6mJ1OyV5acAiOpnX14qRYzdHnXOM9DqGlGpoeeY1Ud4Mo05osOqOUpBJHA9FS
whSZ7VNF+vgnWTLNSYSLI04K1Mdu1nvU6ds+QFz+KkAgMBAAE=
-----END PUBLIC KEY-----

Bob’s RSA public key has the following key identifier:
9eeb67c9b95a74d44d2f16396680e801b5c6a49c

Alice randomly generates a content-encryption key:
c8adc30f4a3e20ac420caca76a68f5787c02ab42afea20d19672fd963a5338e83

Alice randomly generates a key-derivation key:
df85af9e3cebffe6e9b9d24263db31114d0a8e33a0d50e5e64578cc8d1e8b

Alice encrypts the key-derivation key in Bob’s public key:
4e6200431ed95e028f7288daba65d6b90e75959e06888464ac43368f3d978f3d
8179e5837e3c27bf6d166287b99ede69699be77417516ed0790e37c560add0d01
84deb0c917808c8cc720c68d8a9076b6a5e7ecc9093e30fdeaecc9e138d809a
74fcf685f30b2910839551cd8741beedeee6e87c08ff8f30ba87118730c7df7
667002316f1a29a6cc596c77df95a5e389827ed1916bf279299945de080fc7c80
6af62b1a64d9a2n4fa4f1bf4f53e67fca9a8334a27a23500a227d586ee34cbe493
d4a44d40d3e803d54e9e9a71952910dabbedda9a4f310d303331da5c0218d92a
2ef003792251995a9f6c4c403a6f13df1a6265ea70ff7f02d1c6f734264c9a
59196ee8e8df657a028e272ef741eb7711fd5b3f4ea79a9c33df666bf487da710
1c9bbfddaf1c073900a3e9a99da513d8aa332605db07d1c74504cab30c9304a85
d8737f603ec3d3f4056dccc3d756fbeb98245421a4ae151f17ad4e28c5ea077
63358dfb1e5f73435f337b21a38c1a3fa697a530dd97e462f6b5f0252a2d53

Housley [Page 20]
Alice produces a 256-bit key-encryption key with HKDF using SHA-384; the secret value is the key-derivation key; the ‘info’ is the DER-encoded CMSORIforPSKOtherInfo structure with the following values:

```
0   56: SEQUENCE {
  2   32:   OCTET STRING
    :   C2 44 CD D1 1A 0D 1F 39 D9 B6 12 82 77 02 44 FB
    :   0F 6B EF B9 1A B7 F9 6C B0 52 13 36 5C F9 5B 15
  36   1:   ENUMERATED 5
  39  11:   SEQUENCE {
    41   9:     OBJECT IDENTIFIER aes256-wrap
      :     { 2 16 840 1 101 3 4 1 45 }
    :   }
  52   1:   INTEGER 32
  55   1:   INTEGER 32
    :   }
```

The DER encoding of CMSORIforPSKOtherInfo produces 58 octets:

```
30380420c244cdd1a0d1f39d9b61282770244fb0f6befb91ab7f96cb0521336
5cf95b150a0105300b060960864801650304012d020120020120
```

The HKDF output is 256 bits:

```
a14d87451dfdd14d38c49a2adfed3ac49f1d3e62bbdc64ae43b32
```

Alice uses AES-KEY-WRAP to encrypt the 256-bit content-encryption key with the key-encryption key:

```
ae4ea1d9e78fcdcea12d9f10d991ac71502939ee0c30ebdccc97dd1fc5ba3566
c83d0dd5d1b4faa5
```

Alice encrypts the content using AES-256-GCM with the content-encryption key. The 12-octet nonce used is:

```
cafebabe|faced|bad|dec|af888
```

The content plaintext is:

```
48656c6c6f2c20776f726c6421
```

The resulting ciphertext is:

```
9af2d16f21547fceu|ed9b3ef2d
```

The resulting 12-octet authentication tag is:

```
a0e5925cc184e0172463c44c
```
A.2. ContentInfo and AuthEnvelopedData

Alice encodes the AuthEnvelopedData and the ContentInfo, and
sends the result to Bob. The resulting structure is:

0  650: SEQUENCE {
  4  11: OBJECT IDENTIFIER authEnvelopedData
    :   { 1 2 840 113549 1 9 16 1 23 }
  17  633: [0] {
  21  629: SEQUENCE {
  25  1: INTEGER 0
  28  551: SET {
  32  547: [4] {
  36  11: OBJECT IDENTIFIER ** Placeholder **
    :   { 1 2 840 113549 1 9 16 TBD 1 }
  49  530: SEQUENCE {
  53  1: INTEGER 0
  56  19: OCTET STRING 'ptf-kmc:13614122112'
  77  13: SEQUENCE {
  79  11: OBJECT IDENTIFIER ** Placeholder **
    :   { 1 2 840 113549 1 9 16 3 TBD }
    :   }
  92  11: SEQUENCE {
  94  9: OBJECT IDENTIFIER aes256-wrap
    :   { 2 16 840 1 101 3 4 1 45 }
    :   }
 105 432: SEQUENCE {
 109 428: SEQUENCE {
 113  1: INTEGER 2
 116  20: [0]
    :   9E EB 67 C9 B9 5A 74 D4 4D 16 39 66 80 E8 01
    :   B5 CB A4 9C
 138 13: SEQUENCE {
 140  9: OBJECT IDENTIFIER rsaEncryption
    :   { 1 2 840 113549 1 1 1 }
 151  0: NULL
    :   }
 153 384: OCTET STRING
    :   18 09 D6 23 17 DF 02 09 55 57 3B FE 75 95 EB 6A
    :   3D 57 84 6C 69 C1 49 0B F1 11 1A BB 40 0C D8 B5
    :   26 5F D3 62 4B E2 D8 E4 CA EC 6A 12 36 CA 38 E3
    :   A0 7D AA E0 5F A1 E3 BC 59 F3 AD A8 8D 95 A1 6B
    :   06 85 20 93 C7 C5 C0 05 62 ED DF 02 1D FE 68 7C
    :   18 A1 3A AB AA 59 20 6A 1B 92 73 D5 01 C6 5B
    :   FD 1E BB A9 B9 D2 7F 4B 49 7F 3C 4F 3C 13 E3 2B
    :   2A 19 F1 7A CD BC 56 28 EF 7F CA 4F 69 6B 7E 92
    :   66 22 0D 13 B7 23 AD 41 9E 5E 98 2A 80 B7 6C 77
    :   FF 9B 7E B1 04 BA 30 6D 4B 4D F9 25 57 E0 7F 0E

Housley

[Page 22]
: 95 9A 43 6D 14 D5 72 3F AA 8F 66 35 40 D0 E3 71
: 4B 7F 20 9D ED 67 EA 33 79 CD AB 84 16 72 07 D2
: AC 8D 3A DA 12 43 B7 2F 3A CF 91 3E F1 D9 58 20
: 6D F2 9C 09 E1 EC D2 0B 82 BE 5D 69 77 6F FE F7
: EB F6 31 C0 D9 B7 15 BF D0 24 F3 05 1F FF 48 76
: 1D 73 17 19 2C 38 C6 D5 86 BD 67 82 2D B2 61 AA
: 08 C7 E4 37 34 D1 2D E0 51 32 15 4A AC 6B 2B 28
: 5B CD FA 7C 65 89 2F A2 63 DB AB 64 88 43 CC 66
: 27 84 29 AC 15 5F 3B 9E 5B DF 99 AE 4F 1B B2 BC
: 19 6C 17 A1 99 A5 CF F7 80 32 11 88 F1 9D B3 6F
: 4B 16 5F 3F 03 F7 D2 04 3D DE 5F 30 CD 8B BB 3A
: 38 DA 9D EC 16 6C 36 4F 8B 7E 99 AA 99 FB 42 D6
: 1A FF 3C 85 D7 A2 30 74 2C D3 AA F7 18 2A 25 3C
: 5A 02 C4 17 62 21 97 F1 E9 81 83 D0 4E BF 5B 5D

541 40: OCTET STRING
: AE 4E A1 D9 9E 78 FC DC EA 12 D9 F1 0D 99 1A C7
: 15 02 93 9E E0 C3 0E BD CC 97 DD 1F C5 BA 35 66
: C8 3D 0D D5 D1 B4 FA A5

583 55: SEQUENCE {
585 9: OBJECT IDENTIFIER data { 1 2 840 113549 1 7 1 }
596 27: SEQUENCE {
598 9: OBJECT IDENTIFIER aes256-GCM
: { 2 16 840 1 101 3 4 1 46 }
609 14: SEQUENCE {
611 12: OCTET STRING CA FE BA BE FA CE DB AD DE CA F8 88
: }
625 13: [0] 9A F2 D1 6F 21 54 7F CE FE D9 B3 EF 2D
: }
640 12: OCTET STRING A0 E5 92 5C C1 84 E0 17 24 63 C4 4C
: }
: }

Housley
A.3. Recipient Processing Example

Bob’s private key:

```plaintext
BEGIN RSA PRIVATE KEY
MIIG5AIBAAKCAAYEAOcW14xMxcPFJ7F7nEjBZYafC21qapL3ET4jv6C7yGeVRQx
WP Dw1c+cYBBR2ej33j/0ccDmu+XvI2s+5JHKKea+atifuhzas3y1fgeEpeR8+5u
shh20/NblhYRbkh3iAcCgQ56dpDrDvCqVsvS3Jg/VO+OPnZbofc900evt8Q/ro
ahJe1P1y4QudW88zEzij4MlLfbOA9YAYaXyZz2AHHZJeo3nmRnlgJXo6E00E/6q
kjHdHSMSd12W6m6O9TCDZc9y3CAJDU6Ir0vSH7qU18/vN13y40UF6n8hm4km26b
JqzBzt5NnbHy7yUq0QVM3zyEzhr002mrp39aaLNN3HDxXaVtjk759Hzc7ZjaeGW
MjyOf0E3YfEGKrh8fXubj1i76D8UecAzxFFL6m1ji0yV5acAlOpnxl4QyWindH
XOM9DqGiGpeoy1iUu4D0mo5osQqOUpBPBAJ95shx27G7vVnf+vqGWTLNYSYYL04R1Md
uilnvo6s+DQz+bKTAqMBAAECggGATFf1kZkJj4Xczk4aScpSx6+Rakf2h2rS3x
jwghyUfAXTqTEUQ8BSi1HvTHCqXQd+qlXyN3/qa8UEwV4G4NPztY/z5YblwOGJEv
3k8N/ytul6jPFJNop4V0W1bDudTrkMJbXERE6g/rr6dBqeeI7aoCk7N55IHZ0qgh
9xYxU5Bh4rQoCdYLm1t17Tz8CavUg9PQy3vQoDQEowoIjJMV8UQ8qR8HS0k95Jj8AgS
Lq9kxpuvpgjCzoqMCqNePSZVh8x+PFkTRLLrazgLp8STkAtJ6S1l2UzkUqdfDHGK
q/BoXxbDud6L1V2DwDnIS5HxtL54ElcXWsoOyKF8/ilmrHUIURWZFlmSl0k81C5Gx
Udl9rj7VzFTRlyAwmcCEvVR1anshBrhyEyshSOuN5nHJ2WVJ+wShijeKlgElpmk
HrdYiBq4Nz7/xZmIqPhpaAy+yQeanhPB80406C8e7RwKdpaxe44su28F9Eag5qXoU7
8yR1ehGKydX5bbiBLR5cm1VM7zr2BAoHBAP+/e5g2LNF/ECtEBzjeJ0VwshsoUq
haUQPA+9Bx9PytsoKm5QoQhh7QDaxAvrn8/FUW2AaKxaHai9F+/q3OAYSQtEax9j
fdKkKoo3oimN8/yNRSkmhfgjOgj8hd4+GjXqoMSBCEvdT+bajjry8wQqReqRenu
oxU85mb3jv0u1cIKZTIeeyXem5qJQIlMzXSm89B6g71a5EUFly196B0m7hj
/WQzuYYXQdOFZqTkAeEzXNFW21K4w2QKBqWDeiGh41CGTJEcvG7fauMGUuq-
D5dyvMHyf666mx5eS16ejvOr11KzThYyZw8K0wr/CBS2j8ig1GmLrTqrGJ1i
0323050F050mZPueeApOFqgQBDL13JbP6Y/8MhYbsizrVr+Ar4jM0f9M6
W2bF57xh+QFTDMxK6vRCQ6mimBnUZt+ZPs5n/1Y0zAyrgk0aianiy4mjvRly
Vjz6EzG8S1sQcLq/k3Ql1ipSy/oV0rdbJgaw/Aw/UggcEcAgVYGVJKdX0zuvDf9EpV4
mpTBW6yLIV2caKpon/tzi5BgsmeRwv27ytv0mb28px7s/spkq0ubKxz4picy8ucI3
SuY1TAhMH5ux1B3XBySuD2D2r4v1+XM06h6jVRHKhU0nOXDFvngmigP3ZjvY
Boph/j08002Yck4YCTDOXQ8FrbjxSrr+whvRr+kG0gsGcKSVNCPj1N1NISTe4
guj3701muAAzedjn/Vas/PXQovEm0lspPKn9NocrbAbHAJfHJnU22w1/rr
ppmPni1jz130YVcYA5Qv1LKyGaAsnfyp1pLmNnFuVq2j3rsrHx9chHqJ19Hu44PvI-
xc53H0YSFj4ipE3eRmU4q9gy5Wd+1h8fgyuW7E1751sBqEUGvXtrkU5G64T
UR9L11yEyFOP0T1dv/K4Dwryn03tmeVcFADCTkqnsOo1Y3Pcmd270w69gQ
SOey/khdCexfRFa8zuVhm6Cp2ccrzzyBiIR/yCqXOKnQnldQOQKbWbJk5eBFHPfj
AyueKMQESPGYCRrXqQGooVxeArHvKsEdx5wh16JWofYVKV8A0Fy2Myukco/Ex-
2qG5B88D83EqbjT1lq3qrrX2OxtUo88PBF2w1b2LNOwxcbr1yHzeE2BbjyZu51l
sfYI5cP4qJw6s3Q4P4my1W8z2/e1N6VvFmljZYAT7f9WlmtDf7eFVcVzNTvRn6f
hg6GSpJzp4jV3ougi9nQogWZXZ2WiXSX1yPm16Lz34rwoHJtYAA=-
END RSA PRIVATE KEY-----
```

Bob decrypts the key-derivation key with his RSA private key:

```
df85af9e3cebf2de6e89b9d2436d3b31114d0a8e33a0d5e05eb64758ccde81eb
```
Bob produces a 256-bit key-encryption key with HKDF using SHA-384; the secret value is the key-derivation key; the ‘info’ is the DER-encoded CMSORIforPSKOtherInfo structure with the same values as shown in A.1. The HKDF output is 256 bits:
\[ \text{a14d87451dfd11d83cd54ffe2bd38c49a2adfed3ac49f1d3e62bbdc64ae43b32} \]

Bob uses AES-KEY-WRAP to decrypt the content-encryption key with the key-encryption key; the content-encryption key is:
\[ \text{c8adc30f4a3e20ac420caa76a68f5787c02ab42afea20d19672fd963a5338e83} \]

Bob decrypts the content using AES-256-GCM with the content-encryption key, and checks the received authentication tag. The 12-octet nonce used is:
\[ \text{cafebabefacedbaddecaf888} \]

The 12-octet authentication tag is:
\[ \text{a0e5925cc184e0172463c44c} \]

The received ciphertext content is:
\[ \text{9af2d16f21547fcefed9b3ef2d} \]

The resulting plaintext content is:
\[ \text{48656c6c6f2c20776f726c6421} \]

Appendix B: Key Agreement with PSK Example

This example shows the establishment of an AES-256 content-encryption key using:
- a pre-shared key of 256 bits;
- key agreement using ECDH on curve P-384 and X9.63 KDF with SHA-384;
- key derivation using HKDF with SHA-384; and
- key wrap using AES-256-KEYWRAP.

In real-world use, the originator would treat themselves as an additional recipient by performing key agreement with their own static public key and the ephemeral private key generated for this message. This is omitted in an attempt to simplify the example.

B.1. Originator Processing Example

The pre-shared key known to Alice and Bob, in hexadecimal:
\[ \text{4aa53c6f500850dd583a5d9821605c6fa228f5917f87c1c078660214e2d83e4} \]

The identifier assigned to the pre-shared key is:
\[ \text{ptf-kmc:216840110121} \]
Alice randomly generates a content-encryption key:
937b1219a64d57ad81c05cc86017848cc824d4e85800c731c5b7b091033

Alice obtains Bob’s static ECDH public key:
-----BEGIN PUBLIC KEY-----
MHYWwEAYK0ZlZj0CAQYFK4EEACIDYgAEScGFB09nmUWGrgrFGe0Fy9HR/bCo0WyeY
/dePQVwZwmN2yMjmO2diwkCvLlz8U7atinxyIRE9CV54yau1KwW/bkhPDnzuSM
YkcpxMGo32z3JetEloW5aFOja13vW5
-----END PUBLIC KEY-----

It has a key identifier of:
e8218b98b8b4d8675e9ebdc8aeb8c4edcd50529

Alice generates an ephemeral ECDH key pair on the same curve:
-----BEGIN EC PRIVATE KEY-----
MIGkAgEBBDCMiWLQ44ik+L8cVvJrdLCFA+PwlgRF+Wt1Ab25qUh8OB7ePWjxp
/b8P6I0U16GgwYFK4EACKkhZANIAAAQ5G0EmJk/2ks8sXY1kzbuG3U3ttWwQRXAw
LDJICjvYfr+yTqQVkHzm88FAh9MEkw4NKctokKnpsqXyrT3D707600YEnPb
GE51jdxJx9sBzQdAbw1sUOZb7P/7i8=
-----END EC PRIVATE KEY-----

Alice computes a shared secret, called Z, using the Bob’s static
ECDH public key and her ephemeral ECDH private key; Z is:
3f015ed0ff4b99523a95157bbe7e9c0ce52f0c5eb7e41eac79d1c1b6c556
19cf8807e6d800c2de40240e026adc

Alice computes the pairwise key-encryption key, called KEK1, from Z
using the X.9.63 KDF with the ECC-CMS-SharedInfo structure with the
following values:
0  21:  SEQUENCE {
  2  11:  SEQUENCE {
    4  9:  OBJECT IDENTIFIER aes256-wrap
        :   { 2 16 840 1 101 3 4 1 45 }
        :
  }
}
15  6:  [2] {
  17  4:  OCTET STRING 00 00 00 20
      :
  }
}

The DER encoding of ECC-CMS-SharedInfo produces 23 octets:
3015300b06096864801650304012da2060404000000020

The X.9.63 KDF output is the 256-bit KEK1:
27dc25dd1b425f7a968ceada8a0a7f73c6caaab15bafccce4a22a45d6b8f3da
Alice produces the 256-bit KEK2 with HKDF using SHA-384; the secret value is KEK1; the 'info' is the DER-encoded CMSORIforPSKOtherInfo structure with the following values:

0  56: SEQUENCE {
2  32:  OCTET STRING
   :  4A A5 3C BF 50 08 50 DD 58 3A 5D 98 21 60 5C 6F
   :  A2 28 FB 59 17 F8 7C 1C 07 86 60 21 4E 2D 83 E4
36  1:  ENUMERATED 10
39  11:  SEQUENCE {
41  9:   OBJECT IDENTIFIER aes256-wrap
   :   { 2 16 840 1 101 3 4 1 45 }
   : }
52  1:  INTEGER 32
55  1:  INTEGER 32
   : }

The DER encoding of CMSORIforPSKOtherInfo produces 58 octets:
303804204aa53cbf500850dd583a5d9821605c6fa228fb5917f87c1c07866021
4a2d83e40a010a300b060960864801650304012d020120020120

The HKDF output is the 256-bit KEK2:
7de693ee30ae22b5f8f6cd026c2164103f4e1430f1ab135dc1fb98954f9830bb

Alice uses AES-KEY-WRAP to encrypt the content-encryption key with the KEK2; the wrapped key is:
229fe0b45e40003e7d8244ec1b7e7ffb2c8dca16c36f573722553a71263a92b
de0886a602d63f4

Alice encrypts the content using AES-256-GCM with the content-encryption key. The 12-octet nonce used is:
dbaddecaf888cafebabeface

The plaintext is:
48656c6c6f2c20776f726c6421

The resulting ciphertext is:
fc6d6f823e3ed2d209d0c6ffcf

The resulting 12-octet authentication tag is:
550260c42e5b29719426c1ff
B.2. ContentInfo and AuthEnvelopedData

Alice encodes the AuthEnvelopedData and the ContentInfo, and sends the result to Bob. The resulting structure is:

0 327: SEQUENCE {
  4 11: OBJECT IDENTIFIER authEnvelopedData :
    { 1 2 840 113549 1 9 16 1 23 }
  17 310: [0] {
  21 306: SEQUENCE {
  25 1: INTEGER 0
  28 229: SET {
  31 226: [4] {
  34 11: OBJECT IDENTIFIER ** Placeholder ** :
    { 1 2 840 113549 1 9 16 TBD 2 }
  47 210: SEQUENCE {
  50 1: INTEGER 0
  53 20: OCTET STRING ‘ptf-kmc:216840110121’
  75 85: [0] {
  77 83: [1] {
  79 19: SEQUENCE {
  81 6: OBJECT IDENTIFIER :
    dhSinglePass-stdDH-sha256kdf-scheme :
      { 1 3 132 1 11 1 }
  89 9: OBJECT IDENTIFIER aes256-wrap :
    { 2 16 840 1 101 3 4 1 45 }
    }
  100 60: BIT STRING, encapsulates {
  103 57: OCTET STRING :
    1B 41 26 26 4F F6 92 CF 2C 5D BD 64 CD BB 86 DD :
    4B B7 B6 D5 B0 41 15 C0 2C 50 C9 20 28 EF 61 FA :
    FE C9 3A 4E 41 59 1C 86 6F 3C 14 08 7D 30 49 30 :
    E0 D2 9C B6 89 0A 36 0A 6C :
    }
    }
  162 13: SEQUENCE {
  164 11: OBJECT IDENTIFIER ** Placeholder ** :
    { 1 2 840 113549 1 9 16 3 TBD }
    }
  177 11: SEQUENCE {
  179 9: OBJECT IDENTIFIER aes256-wrap :
    { 2 16 840 1 101 3 4 1 45 }
    }
  190 68: SEQUENCE {
  192 66: SEQUENCE {

Housley

[Page 28]
B.3. Recipient Processing Example

Bob obtains Alice’s ephemeral ECDH public key from the message:

-----BEGIN PUBLIC KEY-----
MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEORtBJjZP9pLPLF2NZM27ht11Lt7bVvEEl
wCXQySAo72H6/sk6TkJFZHiZvPBQIftBjMODSnlAdCjYKbKl8q09w7ToO+qCGBDAt
2xhOZSXYz8fbbGdHQAxCJbFNGW+Z/+4v
-----END PUBLIC KEY-----

Bob’s static ECDH private key:

-----BEGIN EC PRIVATE KEY-----
MIGkAgEBBDAnJ4hB+tTUN9X03/W0RsrYy+qcptlRSYkhaDisYQfYzTU0ugjJEEmRk
NTPj4y1IRjegBwWYFK4EEACKhZANiAARjWY8E72eZTAuBsbYSgVj0dH9sKr5j9
149B0wBmbA3b1wnmY733WRYK8tFpxTtq2KfHiH70JXnJq7UpZT/Bu56Ofo51xi
RynEwajfbPc160SWhbIoU6NrxE+/9bk=
-----END EC PRIVATE KEY-----
Bob computes a shared secret, called Z, using the Alice’s ephemeral ECDH public key and his static ECDH private key; Z is:
3f015ed0ff4b99523a95157bbe77e9cc0ee52fcffeb7e41eac79d1c11b6cc556
19cf8807e6d800c2de40240fe0e26adc

Bob computes the pairwise key-encryption key, called KEK1, from Z using the X9.63 KDF with the ECC-CMS-SharedInfo structure with the values shown in B.1. The X9.63 KDF output is the 256-bit KEK1:
27dc25d7b04257a968ceada80a8f73c6ccaab115baafce4a22a45d6b8f3da

Bob produces the 256-bit KEK2 with HKDF using SHA-384; the secret value is KEK1; the ‘info’ is the DER-encoded CMSORIforPSKotherInfo structure with the values shown in B.1. The HKDF output is the 256-bit KEK2:
7de693ee30ae22b5f8f6cd026f2164103f4e1430f1ab135dc1fb98954f9830bb

Bob uses AES-KEY-WRAP to decrypt the content-encryption key with the KEK2; the content-encryption key is:
937b1219a64d57ad81c05cc86075e86017848c824d4e85800c731c5b7b091033

Bob decrypts the content using AES-256-GCM with the content-encryption key, and checks the received authentication tag. The 12-octet nonce used is:
dbaddecaf888cafebabeface

The 12-octet authentication tag is:
550260c42e5b29719426c1ff

The received ciphertext content is:
fc6d6f823e3ed2d209d0c6ffcf

The resulting plaintext content is:
48656c6c6f2c20776f726c6421

Acknowledgements

Many thanks to Roman Danyliw, Ben Kaduk, Burt Kaliski, Panos Kampanakis, Jim Schaad, Robert Sparks, Sean Turner, and Daniel Van Geest for their review and insightful comments. They have greatly improved the design, clarity, and implementation guidance.
Author’s Address

Russ Housley
Vigil Security, LLC
516 Dranesville Road
Herndon, VA 20170
USA
EMail: housley@vigilsec.com
Use of the SHAKE One-way Hash Functions in the Cryptographic Message Syntax (CMS)
draft-ietf-lamps-cms-shakes-17

Abstract

This document updates the "Cryptographic Message Syntax Algorithms" (RFC3370) and describes the conventions for using the SHAKE family of hash functions in the Cryptographic Message Syntax as one-way hash functions with the RSA Probabilistic signature and ECDSA signature algorithms. The conventions for the associated signer public keys in CMS are also described.

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 February 9, 2020.

Copyright Notice

Copyright (c) 2019 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. Change Log .................................................. 2
2. Introduction .................................................. 5
   2.1. Terminology ............................................. 5
3. Identifiers .................................................... 6
4. Use in CMS .................................................... 7
   4.1. Message Digests ........................................... 7
   4.2. Signatures ............................................... 8
      4.2.1. RSASSA-PSS Signatures ................................. 8
      4.2.2. ECDSA Signatures ..................................... 9
   4.3. Public Keys ............................................... 9
   4.4. Message Authentication Codes ............................ 10
5. IANA Considerations ........................................ 10
6. Security Considerations ..................................... 11
7. Acknowledgements ........................................... 11
8. References ................................................... 12
   8.1. Normative References .................................... 12
   8.2. Informative References .................................. 13
Appendix A. ASN.1 Module ...................................... 14
Authors’ Addresses ............................................ 18

1. Change Log

[ EDNOTE: Remove this section before publication. ]

o draft-ietf-lamps-cms-shake-17:
  * Minor updates for EDNOTE accuracy.

o draft-ietf-lamps-cms-shake-16:
  * Minor nits.
  * Using bytes instead of bits for consistency.

o draft-ietf-lamps-cms-shake-15:
  * Minor editorial nits.

o draft-ietf-lamps-cms-shake-14:
  * Fixing error with incorrect preimage resistance bits for SHA128
    and SHA256.
o draft-ietf-lamps-cms-shake-13:
  * Addressing comments from Dan M.’s secdir review.
  * Addressing comment from Scott B.’s opsdir review about references in the abstract.

o draft-ietf-lamps-cms-shake-12:
  * Nits identified by Roman, Barry L. in ballot position review.

o draft-ietf-lamps-cms-shake-11:
  * Minor nits.
  * Nits identified by Roman in AD Review.

o draft-ietf-lamps-cms-shake-10:
  * Updated IANA considerations section to request for OID assignments.

o draft-ietf-lamps-cms-shake-09:
  * Fixed minor text nit.
  * Updates in Sec Considerations section.

o draft-ietf-lamps-cms-shake-08:
  * id-shake128-len and id-shake256-len were replaced with id-shal28 with 32 bytes output length and id-shake256 with 64 bytes output length.
  * Fixed a discrepancy between section 3 and 4.4 about the KMAC OIDs that have parameters as optional.

o draft-ietf-lamps-cms-shake-07:
  * Small nit from Russ while in WGLC.

o draft-ietf-lamps-cms-shake-06:
  * Incorporated Eric’s suggestion from WGLC.

o draft-ietf-lamps-cms-shake-05:
  * Added informative references.
* Updated ASN.1 so it compiles.
* Updated IANA considerations.

o draft-ietf-lamps-cms-shake-04:
  * Added RFC8174 reference and text.
  * Explicitly explained why RSASSA-PSS-params are omitted in section 4.2.1.
  * Simplified Public Keys section by removing redundant info from RFCs.

o draft-ietf-lamps-cms-shake-03:
  * Removed paragraph suggesting KMAC to be used in generating k in Deterministic ECDSA. That should be RFC6979-bis.
  * Removed paragraph from Security Considerations that talks about randomness of k because we are using deterministic ECDSA.
  * Completed ASN.1 module and fixed KMAC ASN.1 based on Jim’s feedback.
  * Text fixes.

o draft-ietf-lamps-cms-shake-02:
  * Updates based on suggestions and clarifications by Jim.
  * Started ASN.1 module.

o draft-ietf-lamps-cms-shake-01:
  * Significant reorganization of the sections to simplify the introduction, the new OIDs and their use in CMS.
  * Added new OIDs for RSASSA-PSS that hardcodes hash, salt and MGF, according the WG consensus.
  * Updated Public Key section to use the new RSASSA-PSS OIDs and clarify the algorithm identifier usage.
  * Removed the no longer used SHAKE OIDs from section 3.1.

o draft-ietf-lamps-cms-shake-00:
2. Introduction

The "Cryptographic Message Syntax (CMS)" [RFC5652] is used to digitally sign, digest, authenticate, or encrypt arbitrary message contents. "Cryptographic Message Syntax (CMS) Algorithms" [RFC3370] defines the use of common cryptographic algorithms with CMS. This specification updates RFC3370 and describes the use of the SHAKE128 and SHAKE256 specified in [SHA3] as new hash functions in CMS. In addition, it describes the use of these functions with the RSASSA-PSS signature algorithm [RFC8017] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [X9.62] with the CMS signed-data content type.

In the SHA-3 family, two extendable-output functions (SHAKEs), SHAKE128 and SHAKE256, are defined. Four other hash function instances, SHA3-224, SHA3-256, SHA3-384, and SHA3-512, are also defined but are out of scope for this document. A SHAKE is a variable length hash function defined as SHAKE(M, d) where the output is a d-bits-long digest of message M. The corresponding collision and second-preimage-resistance strengths for SHAKE128 are \( \min(d/2,128) \) and \( \min(d,128) \) bits, respectively (Appendix A.1 [SHA3]). And the corresponding collision and second-preimage-resistance strengths for SHAKE256 are \( \min(d/2,256) \) and \( \min(d,256) \) bits, respectively. In this specification we use \( d=256 \) (for SHAKE128) and \( d=512 \) (for SHAKE256).

A SHAKE can be used in CMS as the message digest function (to hash the message to be signed) in RSASSA-PSS and ECDSA, message authentication code and as the mask generation function (MGF) in RSASSA-PSS. This specification describes the identifiers for SHAKEs to be used in CMS and their meaning.

2.1. Terminology

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.
3. Identifiers

This section identifies eight new object identifiers (OIDs) for using SHAKE128 and SHAKE256 in CMS.

Two object identifiers for SHAKE128 and SHAKE256 hash functions are defined in [shake-nist-oids] and we include them here for convenience.

id-shake128 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
country(16) us(840) organization(1) gov(101) csor(3)
nistAlgorithm(4) 2 11 }

id-shake256 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
country(16) us(840) organization(1) gov(101) csor(3)
nistAlgorithm(4) 2 12 }

In this specification, when using the id-shake128 or id-shake256 algorithm identifiers, the parameters MUST be absent. That is, the identifier SHALL be a SEQUENCE of one component, the OID.

[I-D.ietf-lamps-pkix-shake] [ EDNOTE: Update reference with the RFC when it is published. ] defines two identifiers for RSASSA-PSS signatures using SHAKEs which we include here for convenience. [ EDNOTE: Update the TBD1-2 reference when the RFC (ietf-lamps-pkix-shake) is published. ]

id-RSASSA-PSS-SHAKE128 OBJECT IDENTIFIER ::= { iso(1)
identified-organization(3) dod(6) internet(1)
security(5) mechanisms(5) pkix(7) algorithms(6)
TBD1 }

id-RSASSA-PSS-SHAKE256 OBJECT IDENTIFIER ::= { iso(1)
identified-organization(3) dod(6) internet(1)
security(5) mechanisms(5) pkix(7) algorithms(6)
TBD2 }

The same RSASSA-PSS algorithm identifiers can be used for identifying public keys and signatures.

[I-D.ietf-lamps-pkix-shake] [ EDNOTE: Update reference with the RFC when it is published. ] also defines two algorithm identifiers of ECDSA signatures using SHAKEs which we include here for convenience. [ EDNOTE: Update the TBD3-4 reference when the RFC (ietf-lamps-pkix-shake) is published. ]
id-ecdsa-with-shake128 OBJECT IDENTIFIER ::= { iso(1)
  identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) algorithms(6)
  TBD3 }

id-ecdsa-with-shake256 OBJECT IDENTIFIER ::= { iso(1)
  identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) algorithms(6)
  TBD4 }

The parameters for the four RSASSA-PSS and ECDSA identifiers MUST be
absent. That is, each identifier SHALL be a SEQUENCE of one
component, the OID.

Two object identifiers for KMACs using SHAKE128 and SHAKE256 as
defined in by the National Institute of Standards and Technology
(NIST) in [shake-nist-oids] and we include them here for convenience.

id-KmacWithSHAKE128 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
  country(16) us(840) organization(1) gov(101) csor(3)
  nistAlgorithm(4) 2 19 }

id-KmacWithSHAKE256 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
  country(16) us(840) organization(1) gov(101) csor(3)
  nistAlgorithm(4) 2 20 }

The parameters for id-KmacWithSHAKE128 and id-KmacWithSHAKE256 are
OPTIONAL.

Section 4.1, Section 4.2.1, Section 4.2.2 and Section 4.4 specify the
required output length for each use of SHAKE128 or SHAKE256 in
message digests, RSASSA-PSS, ECDSA and KMAC.

4. Use in CMS

4.1. Message Digests

The id-shake128 and id-shake256 OIDs (Section 3) can be used as the
digest algorithm identifiers located in the SignedData, SignerInfo,
DigestedData, and the AuthenticatedData digestAlgorithm fields in CMS
[RFC5652]. The OID encoding MUST omit the parameters field and the
output length of SHAKE128 or SHAKE256 as the message digest MUST be
32 or 64 bytes, respectively.

The digest values are located in the DigestedData field and the
Message Digest authenticated attribute included in the
signedAttributes of the SignedData signerInfo. In addition, digest
values are input to signature algorithms. The digest algorithm MUST be the same as the message hash algorithms used in signatures.

4.2. Signatures

In CMS, signature algorithm identifiers are located in the SignerInfo signatureAlgorithm field of SignedData content type and countersignature attribute. Signature values are located in the SignerInfo signature field of SignedData content type and countersignature attribute.

Conforming implementations that process RSASSA-PSS and ECDSA with SHAKE signatures when processing CMS data MUST recognize the corresponding OIDs specified in Section 3.

When using RSASSA-PSS or ECDSA with SHAKEs, the RSA modulus or ECDSA curve order SHOULD be chosen in line with the SHAKE output length. Refer to Section 6 for more details.

4.2.1. RSASSA-PSS Signatures

The RSASSA-PSS algorithm is defined in [RFC8017]. When id-RSASSA-PSS-SHAKE128 or id-RSASSA-PSS-SHAKE256 specified in Section 3 is used, the encoding MUST omit the parameters field. That is, the AlgorithmIdentifier SHALL be a SEQUENCE of one component, id-RSASSA-PSS-SHAKE128 or id-RSASSA-PSS-SHAKE256. [RFC4055] defines RSASSA-PSS-params that are used to define the algorithms and inputs to the algorithm. This specification does not use parameters because the hash, mask generation function, trailer and salt are embedded in the OID definition.

The hash algorithm to hash a message being signed and the hash algorithm as the mask generation function used in RSASSA-PSS MUST be the same: both SHAKE128 or both SHAKE256. The output length of the hash algorithm which hashes the message SHALL be 32 (for SHAKE128) or 64 bytes (for SHAKE256).

The mask generation function takes an octet string of variable length and a desired output length as input, and outputs an octet string of the desired length. In RSASSA-PSS with SHAKEs, the SHAKEs MUST be used natively as the MGF function, instead of the MGF1 algorithm that uses the hash function in multiple iterations as specified in Section B.2.1 of [RFC8017]. In other words, the MGF is defined as the SHAKE128 or SHAKE256 with input being the mgfSeed for id-RSASSA-PSS-SHAKE128 and id-RSASSA-PSS-SHAKE256, respectively. The mgfSeed is the seed from which mask is generated, an octet string [RFC8017]. As explained in Step 9 of section 9.1.1 of [RFC8017], the output length of the MGF is emLen - hLen - 1 bytes. emLen is the maximum
message length $\lceil(n-1)/8\rceil$, where $n$ is the RSA modulus in bits. hLen is 32 and 64-bytes for id-RSASSA-PSS-SHAKE128 and id-RSASSA-PSS-SHAKE256, respectively. Thus when SHAKE is used as the MGF, the SHAKE output length maskLen is $(8\cdot\text{emLen} - 264)$ or $(8\cdot\text{emLen} - 520)$ bits, respectively. For example, when RSA modulus $n$ is 2048, the output length of SHAKE128 or SHAKE256 as the MGF will be 1784 or 1528-bits when id-RSASSA-PSS-SHAKE128 or id-RSASSA-PSS-SHAKE256 is used, respectively.

The RSASSA-PSS saltLength MUST be 32 bytes for id-RSASSA-PSS-SHAKE128 or 64 bytes for id-RSASSA-PSS-SHAKE256. Finally, the trailerField MUST be 1, which represents the trailer field with hexadecimal value $0x\text{BC}$ [RFC8017].

4.2.2. ECDSA Signatures

The Elliptic Curve Digital Signature Algorithm (ECDSA) is defined in [X9.62]. When the id-ecdsa-with-shake128 or id-ecdsa-with-shake256 (specified in Section 3) algorithm identifier appears, the respective SHAKE function is used as the hash. The encoding MUST omit the parameters field. That is, the AlgorithmIdentifier SHALL be a SEQUENCE of one component, the OID id-ecdsa-with-shake128 or id-ecdsa-with-shake256.

For simplicity and compliance with the ECDSA standard specification, the output length of the hash function must be explicitly determined. The output length for SHAKE128 or SHAKE256 used in ECDSA MUST be 32 or 64 bytes, respectively.

Conforming CA implementations that generate ECDSA with SHAKE signatures in certificates or CRLs SHOULD generate such signatures with a deterministically generated, non-random $k$ in accordance with all the requirements specified in [RFC6979]. They MAY also generate such signatures in accordance with all other recommendations in [X9.62] or [SEC1] if they have a stated policy that requires conformance to those standards. Those standards have not specified SHAKE128 and SHAKE256 as hash algorithm options. However, SHAKE128 and SHAKE256 with output length being 32 and 64 octets, respectively can be used instead of 256 and 512-bit output hash algorithms such as SHA256 and SHA512.

4.3. Public Keys

In CMS, the signer’s public key algorithm identifiers are located in the OriginatorPublicKey’s algorithm attribute. The conventions and encoding for RSASSA-PSS and ECDSA public keys algorithm identifiers are as specified in Section 2.3 of [RFC3279], Section 3.1 of [RFC4055] and Section 2.1 of [RFC5480].
Traditionally, the rsaEncryption object identifier is used to identify RSA public keys. The rsaEncryption object identifier continues to identify the public key when the RSA private key owner does not wish to limit the use of the public key exclusively to RSASSA-PSS with SHAKEs. When the RSA private key owner wishes to limit the use of the public key exclusively to RSASSA-PSS, the AlgorithmIdentifier for RSASSA-PSS defined in Section 3 SHOULD be used as the algorithm attribute in the OriginatorPublicKey sequence. Conforming client implementations that process RSASSA-PSS with SHAKE public keys in CMS message MUST recognize the corresponding OIDs in Section 3.

Conforming implementations MUST specify and process the algorithms explicitly by using the OIDs specified in Section 3 when encoding ECDSA with SHAKE public keys in CMS messages.

The identifier parameters, as explained in Section 3, MUST be absent.

4.4. Message Authentication Codes

KMAC message authentication code (KMAC) is specified in [SP800-185]. In CMS, KMAC algorithm identifiers are located in the AuthenticatedData macAlgorithm field. The KMAC values are located in the AuthenticatedData mac field.

When the id-KmacWithSHAKE128 or id-KmacWithSHAKE256 OID is used as the MAC algorithm identifier, the parameters field is optional (absent or present). If absent, the SHAKE256 output length used in KMAC is 32 or 64 bytes, respectively, and the customization string is an empty string by default.

Conforming implementations that process KMACs with the SHAKEs when processing CMS data MUST recognize these identifiers.

When calculating the KMAC output, the variable N is 0xD2B282C2, S is an empty string, and L, the integer representing the requested output length in bits, is 256 or 512 for KmacWithSHAKE128 or KmacWithSHAKE256, respectively, in this specification.

5. IANA Considerations

One object identifier for the ASN.1 module in Appendix A was requested for the SMI Security for S/MIME Module Identifiers (1.2.840.113549.1.9.16.0) registry:
6. Security Considerations

This document updates [RFC3370]. The security considerations section of that document applies to this specification as well.

NIST has defined appropriate use of the hash functions in terms of the algorithm strengths and expected time frames for secure use in Special Publications (SPs) [SP800-78-4] and [SP800-107]. These documents can be used as guides to choose appropriate key sizes for various security scenarios.

SHAKE128 with output length of 32 bytes offers 128-bits of collision and preimage resistance. Thus, SHAKE128 OIDs in this specification are RECOMMENDED with 2048 (112-bit security) or 3072-bit (128-bit security) RSA modulus or curves with group order of 256-bits (128-bit security). SHAKE256 with 64 bytes output length offers 256-bits of collision and preimage resistance. Thus, the SHAKE256 OIDs in this specification are RECOMMENDED with 4096-bit RSA modulus or higher or curves with group order of at least 512 bits such as NIST Curve P-521 (256-bit security). Note that we recommended 4096-bit RSA because we would need 15360-bit modulus for 256-bits of security which is impractical for today’s technology.

When more than two parties share the same message-authentication key, data origin authentication is not provided. Any party that knows the message-authentication key can compute a valid MAC, therefore the content could originate from any one of the parties.

7. Acknowledgements

This document is based on Russ Housley’s draft [I-D.housley-lamps-cms-sha3-hash]. It replaces SHA3 hash functions by SHAKE128 and SHAKE256 as the LAMPS WG agreed.

The authors would like to thank Russ Housley for his guidance and very valuable contributions with the ASN.1 module. Valuable feedback was also provided by Eric Rescorla.
8. References

8.1. Normative References


8.2. Informative References

[I-D.housley-lamps-cms-sha3-hash]
Housley, R., "Use of the SHA3 One-way Hash Functions in the Cryptographic Message Syntax (CMS)", draft-housley-lamps-cms-sha3-hash-00 (work in progress), March 2017.

[I-D.ietf-lamps-pkix-shake]


Kampanakis & Dang Expires February 9, 2020 [Page 13]


Appendix A. ASN.1 Module

This appendix includes the ASN.1 modules for SHAKEs in CMS. This module includes some ASN.1 from other standards for reference.

CMSAlgsForSHAKE-2019 { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0) id-mod-cms-shakes-2019(TBD) }

DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS ALL;

IMPORTS

DIGEST-ALGORITHM, MAC-ALGORITHM, SMIME-CAPS FROM AlgorithmInformation-2009
-- Message Digest Algorithms (mda-)
-- used in SignedData, SignerInfo, DigestedData,
-- and the AuthenticatedData digestAlgorithm
-- fields in CMS
--
MessageDigestAlgs DIGEST-ALGORITHM ::= {
    -- This expands MessageAuthAlgs from [RFC5652]
    -- and MessageDigestAlgs in [RFC5753]
    mda-shake128 |
    mda-shake256,
    ...}

-- One-Way Hash Functions
-- SHAKE128
mda-shake128 DIGEST-ALGORITHM ::= {
    IDENTIFIER id-shake128  -- with output length 32 bytes.
}
id-shake128 OBJECT IDENTIFIER ::= {
    joint-iso-itu-t(2) country(16)
    us(840) organization(1) gov(101)
    csor(3) nistAlgorithm(4)
    hashAlgs(2) 11 } 

-- SHAKE256
mda-shake256 DIGEST-ALGORITHM ::= {
    IDENTIFIER id-shake256  -- with output length 64 bytes.
}
id-shake256 OBJECT IDENTIFIER ::= {
    joint-iso-itu-t(2) country(16)
    us(840) organization(1) gov(101)
    csor(3) nistAlgorithm(4)
    hashAlgs(2) 12 } 

-- Public key algorithm identifiers located in the
-- OriginatorPublicKey’s algorithm attribute in CMS.
-- And Signature identifiers used in SignerInfo
-- signatureAlgorithm field of SignedData content
-- type and countersignature attribute in CMS.
--
-- From RFC5280, for reference.
-- rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }
--   When the rsaEncryption algorithm identifier is used
--     for a public key, the AlgorithmIdentifier parameters
--     field MUST contain NULL.
--
-- id-RSASSA-PSS-SHAKE128 OBJECT IDENTIFIER ::= { iso(1)
--     identified-organization(3) dod(6) internet(1)
--     security(5) mechanisms(5) pkix(7) algorithms(6)
--     TBD1 }
-- id-RSASSA-PSS-SHAKE256 OBJECT IDENTIFIER ::= { iso(1)
--     identified-organization(3) dod(6) internet(1)
--     security(5) mechanisms(5) pkix(7) algorithms(6)
--     TBD2 }
--   When the id-RSASSA-PSS-* algorithm identifiers are used
--     for a public key or signature in CMS, the AlgorithmIdentifier
--     parameters field MUST be absent. The message digest algorithm
--     used in RSASSA-PSS MUST be SHAKE128 or SHAKE256 with a 32 or
--     64 byte output length, respectively. The mask generation
--     function MUST be SHAKE128 or SHAKE256 with an output length
--     of (8*ceil((n-1)/8) - 264) or (8*ceil((n-1)/8) - 520) bits,
--     respectively, where n is the RSA modulus in bits.
--     The RSASSA-PSS salt length MUST be 32 or 64 bytes, respectively.
--     The trailerField MUST be 1, which represents the trailer
--     field with hexadecimal value 0xBC. Regardless of
--     id-RSASSA-PSS-* or rsaEncryption being used as the
--     AlgorithmIdentifier of the OriginatorPublicKey, the RSA
--     public key MUST be encoded using the RSAPublicKey type.

-- From RFC4055, for reference.
-- RSAPublicKey ::= SEQUENCE {
--   modulus INTEGER, -- n
--   publicExponent INTEGER }

-- id-ecdsa-with-shake128 OBJECT IDENTIFIER ::= { iso(1)
--     identified-organization(3) dod(6) internet(1)
--     security(5) mechanisms(5) pkix(7) algorithms(6)
--     TBD3 }
-- id-ecdsa-with-shake256 OBJECT IDENTIFIER ::= { iso(1)
--     identified-organization(3) dod(6) internet(1)
--     security(5) mechanisms(5) pkix(7) algorithms(6)
--     TBD4 }
--   When the id-ecdsa-with-shake* algorithm identifiers are
--   used in CMS, the AlgorithmIdentifier parameters field
--   MUST be absent and the signature algorithm should be
--   deterministic ECDSA [RFC6979]. The message digest MUST
-- be SHAKE128 or SHAKE256 with a 32 or 64 byte output
-- length, respectively. In both cases, the ECDSA public key,
-- MUST be encoded using the id-ecPublicKey type.

-- From RFC5480, for reference.
-- id-ecPublicKey OBJECT IDENTIFIER ::= {  
--   iso(1) member-body(2) us(840) ansi-X9-62(10045) keyType(2) 1 }
-- The id-ecPublicKey parameters must be absent or present
-- and are defined as
-- ECPParameters ::= CHOICE {
--   namedCurve OBJECT IDENTIFIER
--   -- implicitCurve NULL
--   -- specifiedCurve SpecifiedECDomain
-- }

-- Message Authentication (maca-) Algorithms
-- used in AuthenticatedData macAlgorithm in CMS
-- MessageAuthAlgs MAC-ALGORITHM ::= {
--   -- This expands MessageAuthAlgs from [RFC5652] and [RFC6268]
-- maca-KMACwithSHAKE128 |
-- maca-KMACwithSHAKE256,
--    ... }

SMimeCaps SMIME-CAPS ::= {
-- The expands SMimeCaps from [RFC5911]
-- maca-KMACwithSHAKE128&smimeCaps |
-- maca-KMACwithSHAKE256&smimeCaps,
--    ... }

-- KMAC with SHAKE128
-- maca-KMACwithSHAKE128 MAC-ALGORITHM ::= {
 IDENTIFIER id-KMACWithSHAKE128
PARAMS TYPE KMACwithSHAKE128-params ARE optional
-- If KMACwithSHAKE128-params parameters are absent
-- the SHAKE128 output length used in KMAC is 256 bits
-- and the customization string is an empty string.
IS-KEYED-MAC TRUE
SMIME-CAPS (IDENTIFIED BY id-KMACWithSHAKE128)
}

id-KMACWithSHAKE128 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
country(16) us(840) organization(1)
gov(101) csor(3) nistAlgorithm(4)hashAlgs(2) 19 }
KMACwithSHAKE128-params ::= SEQUENCE {
  kMACOutputLength INTEGER DEFAULT 256, -- Output length in bits
  customizationString OCTET STRING DEFAULT ''H
}

-- KMAC with SHAKE256
maca-KMACwithSHAKE256 MAC-ALGORITHM ::= {
  IDENTIFIER id-KMACWithSHAKE256
  PARAMS TYPE KMACwithSHAKE256-params ARE optional
    -- If KMACwithSHAKE256-params parameters are absent
    -- the SHAKE256 output length used in KMAC is 512 bits
    -- and the customization string is an empty string.
    IS-KEYED-MAC TRUE
    SMIME-CAPS {IDENTIFIED BY id-KMACWithSHAKE256}
}

id-KMACWithSHAKE256 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
    country(16) us(840) organization(1)
    gov(101) csor(3) nistAlgorithm(4)
    hashAlgs(2) 20 }

KMACwithSHAKE256-params ::= SEQUENCE {
  kMACOutputLength INTEGER DEFAULT 512, -- Output length in bits
  customizationString OCTET STRING DEFAULT ''H
}

END

Authors’ Addresses

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Quynh Dang
NIST
100 Bureau Drive
Gaithersburg, MD 20899

Email: quynh.Dang@nist.gov
Hash Of Root Key Certificate Extension
draft-ietf-lamps-hash-of-root-key-cert-extn-07

Abstract

This document specifies the Hash Of Root Key certificate extension. This
certificate extension is carried in the self-signed certificate for a trust anchor,
which is often called a Root Certification Authority (CA) certificate. This
certificate extension unambiguously identifies the next public key that will be used
at some point in the future as the next Root CA certificate, eventually replacing
the current one.

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 December 31, 2019.

Copyright Notice

Copyright (c) 2019 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
1. Introduction

This document specifies the Hash Of Root Key X.509 version 3 certificate extension. The extension is an optional addition to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile [RFC5280]. The certificate extension facilitates the orderly transition from one Root Certification Authority (CA) public key to the next. It does so by publishing the hash value of the next generation public key in the current self-signed certificate. This hash value is a commitment to a particular public key in the next generation self-signed certificate. This commitment allows a relying party to unambiguously recognize the next generation self-signed certificate when it becomes available, install the new self-signed certificate in the trust anchor store, and eventually remove the previous one from the trust anchor store.

A Root CA Certificate MAY include the Hashed Root Key certificate extension to provide the hash value of the next public key that will be used by the Root CA.

1.1. Terminology

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.
1.2. ASN.1

Certificates [RFC5280] use ASN.1 [X680]; Distinguished Encoding Rules (DER) [X690] are REQUIRED for certificate signing and validation.

2. Overview

Before the initial deployment of the Root CA, the following are generated:

- \( R_1 = \) The initial Root key pair
- \( R_2 = \) The second generation Root key pair
- \( H_2 = \) Thumbprint (hash) of the public key of \( R_2 \)
- \( C_1 = \) Self-signed certificate for \( R_1 \), which also contains \( H_2 \)

\( C_1 \) is a self-signed certificate, and it contains \( H_2 \) within the HashOfRootKey extension. \( C_1 \) is distributed as part of the initial system deployment. The HashOfRootKey certificate extension is described in Section 3.

When the time comes to replace the initial Root CA certificate, \( R_1 \), the following are generated:

- \( R_3 = \) The third generation Root key pair
- \( H_3 = \) Thumbprint (hash) the public key of \( R_3 \)
- \( C_2 = \) Self-signed certificate for \( R_2 \), which contains \( H_3 \)

This is an iterative process. That is, \( R_4 \) and \( H_4 \) are generated when it is time for \( C_3 \) to replace \( C_2 \). And so on.

The successor to the Root CA self-signed certificate can be delivered by any means. Whenever a new Root CA self-signed certificate is received, the recipient is able to verify that the potential Root CA certificate links back to a previously authenticated Root CA certificate with the hashOfRootKey certificate extension. That is, the recipient verifies the signature on the self-signed certificate and verifies that the hash of the DER-encoded SubjectPublicKeyInfo from the potential Root CA certificate matches the value from the HashOfRootKey certificate extension of the current Root CA certificate. Checking the self-signed certificate signature ensures that the certificate contains the subject name, public key algorithm identifier, and public key algorithm parameters intended by the key owner; these are important inputs to certification path validation as defined in Section 6 of [RFC5280]. Checking the hash of the SubjectPublicKeyInfo ensures that the certificate contains the intended public key. If either check fails, then the potential Root CA certificate is not a valid replacement, and the recipient continues to use the current Root CA certificate. If both checks
succeed, then the recipient adds the potential Root CA certificate to
the trust anchor store. As discussed in Section 5, the recipient can
remove the current Root CA certificate immediately in some
situations. In other situations, the recipient waits an appropriate
amount of time to ensure that existing certification paths continue
to validate.

3. Hash Of Root Key Certificate Extension

The HashOfRootKey certificate extension MUST NOT be critical.

The following ASN.1 [X680][X690] syntax defines the HashOfRootKey
certificate extension:

\[
\text{ext-HashOfRootKey EXTENSION ::= } \{ \text{SYNTAX HashedRootKey, IDENTIFIED BY id-ce-hashOfRootKey, CRITICALITY } \{ \text{FALSE} \} \}
\]

\[
\text{HashedRootKey ::= SEQUENCE } \{ \text{hashAlg HashAlgorithm, hashValue OCTET STRING } \}
\]

\[
\text{id-ce-hashOfRootKey ::= OBJECT IDENTIFIER } \{ 1 3 6 1 4 1 51483 2 1 \}
\]

The definitions of EXTENSION and HashAlgorithm can be found in
[RFC5912].

The hashAlg indicates the one-way hash algorithm that was used to
compute the hash value.

The hashValue contains the hash value computed from the next
generation public key. The public key is DER-encoded
SubjectPublicKeyInfo as defined in [RFC5280].

4. IANA Considerations

This document makes no requests of the IANA.

5. Operational Considerations

Guidance on the transition from one trust anchor to another is
available in Section 4.4 of [RFC4210]. In particular, the oldWithNew
and newWithOld advice ensures that relying parties are able to
validate certificates issued under the current Root CA certificate
and the next generation Root CA certificate throughout the
transition. The notAfter field in the oldWithNew certificate MUST
cover the validity period of all unexpired certificates issued under the old Root CA private key. Further, this advice SHOULD be followed by Root CAs to avoid the need for all relying parties to make the transition at the same time.

After issuing the newWithOld certificate, the Root CA MUST stop using the old private key to sign certificates.

Some enterprise and application-specific environments offer a directory service or certificate repository to make certificate and CRLs available to relying parties. Section 3 in [RFC5280] describes a certificate repository. When a certificate repository is available, the oldWithNew and newWithOld certificates SHOULD be published before the successor to the current Root CA self-signed certificate is released. Recipients that are able to obtain the oldWithNew certificate SHOULD immediately remove the old Root CA self-signed certificate from the trust anchor store.

In environments without such a directory service or repository, like the Web PKI, recipients need a way to obtain the oldWithNew and newWithOld certificates. The Root CA SHOULD include the subject information access extension [RFC5280] with the accessMethod set to id-ad-caRepository and the assessLocation set to the HTTP URL that can be used to fetch a DER-encoded "certs-only" (simple PKI response) message as specified in [RFC5272] in all of their self-signed certificates. The Root CA SHOULD publish the "certs-only" message with the oldWithNew certificate and the newWithOld certificate before the subsequent Root CA self-signed certificate is released. The "certs-only" message format allows certificates to be added and removed from the bag of certificates over time, so the same HTTP URL can be used throughout the lifetime of the Root CA.

In environments without such a directory service or repository, recipients SHOULD keep both the old and replacement Root CA self-signed certificates in the trust anchor store for some amount of time to ensure that all end-entity certificates can be validated until they expire. The recipient MAY keep the old Root CA self-signed certificate until all of the certificates in the local cache that are subordinate to it have expired.

Certification path construction is more complex when the trust anchor store contains multiple self-signed certificates with the same distinguished name. For this reason, the replacement Root CA self-signed certificate SHOULD contain a different distinguished name than the one it is replacing. One approach is to include a number as part of the name that is incremented with each generation, such as "Example CA", "Example CA G2", "Example CA G3", and so on.
Changing names from one generation to another can lead to confusion when reviewing the history of a trust anchor store. To assist with such review, a recipient MAY create an audit entry to capture the old and replacement self-signed certificates.

The Root CA must securely back up the yet-to-be-deployed key pair. If the Root CA stores the key pair in a hardware security module, and that module fails, the Root CA remains committed to the key pair that is no longer available. This leaves the Root CA with no alternative but to deploy a new self-signed certificate that contains a newly-generated key pair in the same manner as the initial self-signed certificate, thus losing the benefits of the Hash Of Root Key certificate extension altogether.

6. Security Considerations

The security considerations from [RFC5280] apply, especially the discussion of self-issued certificates.

The Hash Of Root Key certificate extension facilitates the orderly transition from one Root CA public key to the next by publishing the hash value of the next generation public key in the current certificate. This allows a relying party to unambiguously recognize the next generation public key when it becomes available; however, the full public key is not disclosed until the Root CA releases the next generation certificate. In this way, attackers cannot begin to analyze the public key before the next generation Root CA self-signed certificate is released.

The Root CA needs to ensure that the public key in the next generation certificate is as strong or stronger than the key that it is replacing. Of course, a significant advance in cryptoanalytic capability can break the yet-to-be-deployed key pair. Such advances are rare and difficult to predict. If such an advance occurs, the Root CA remains committed to the now broken key. This leaves the Root CA with no alternative but to deploy a new self-signed certificate that contains a newly-generated key pair, most likely using a different signature algorithm, in the same manner as the initial self-signed certificate, thus losing the benefits of the Hash Of Root Key certificate extension altogether.

The Root CA needs to employ a hash function that is resistant to preimage attacks [RFC4270]. A first-preimage attack against the hash function would allow an attacker to find another input that results published hash value. For the attack to be successful, the input would have to be a valid SubjectPublicKeyInfo that contains a public key that corresponds to a private key known to the attacker. A second-preimage attack becomes possible once the Root CA releases the
next generation public key, which makes the input to the hash function available to the attacker and everyone else. Again, the attacker needs to find a valid SubjectPublicKeyInfo that contains the public key that corresponds to a private key known to the attacker. If the employed hash function is broken after the Root CA publishes the self-signed certificate with the HashOfRootKey certificate extension, an attacker would be able to trick the recipient into installing the incorrect next generation certificate in the trust anchor store.

If an early release of the next generation public key occurs and the Root CA is concerned that attackers were given too much lead time to analyze that public key, then the Root CA can transition to a freshly generated key pair by rapidly performing two transitions. The first transition takes the Root CA to the key pair that suffered the early release, and it causes the Root CA to generate the subsequent Root key pair. The second transition occurs when the Root CA is confident that the population of relying parties have completed the first transition, and it takes the Root CA to the freshly generated key pair. Of course, the second transition also causes the Root CA to generate another key pair that is reserved for future use. Queries for the CRLs associated with certificates that are subordinate to the self-signed certificate can give some indication for the number of relying parties that are still actively using the self-signed certificates.

7. Acknowledgements

The Secure Electronic Transaction (SET) [SET] specification published by MasterCard and VISA in 1997 includes a very similar certificate extension. The SET certificate extension has essentially the same semantics, but the syntax fairly different.

CTIA - The Wireless Association - is developing a public key infrastructure that will make use of the certificate extension described in this document, and the object identifiers used in the ASN.1 module were assigned by CTIA.

Many thanks to Stefan Santesson, Jim Schaad, Daniel Kahn Gillmor, Joel Halpern, Paul Hoffman, Rich Salz, and Ben Kaduk. Their review and comments have greatly improved the document, especially the Operational Considerations and Security Considerations sections.

8. References
8.1. Normative References


Appendix A.  ASN.1 Module

The following ASN.1 module provides the complete definition of the HashOfRootKey certificate extension.
HashedRootKeyCertExtn { 1 3 6 1 4 1 51483 0 1 }

DEFINITIONS IMPLICIT TAGS ::= BEGIN

-- EXPORTS All IMPORTS

HashAlgorithm
    FROM PKIX1-PSS-OAEP-Algorithms-2009 -- [RFC5912]
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-rsa-pkalgs-02(54) }

EXTENSION
    FROM PKIX-CommonTypes-2009
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkixCommon-02(57) } ;

--
-- Expand the certificate extensions list in [RFC5912]
--

CertExtensions EXTENSION ::= {
    ext-HashOfRootKey, ... }

--
-- HashOfRootKey Certificate Extension
--

ext-HashOfRootKey EXTENSION ::= {
    SYNTAX HashedRootKey -- Only in Root CA certificates
    IDENTIFIED BY id-ce-hashOfRootKey
    CRITICALITY {FALSE} }

HashedRootKey ::= SEQUENCE {
    hashAlg HashAlgorithm, -- Hash algorithm used
    hashValue OCTET STRING -- Hash of DER-encoded
    SubjectPublicKeyInfo
}

id-ce-hashOfRootKey OBJECT IDENTIFIER ::= { 1 3 6 1 4 1 51483 2 1 }

END
Author’s Address

Russ Housley
Vigil Security
516 Dranesville Road
Herndon, VA  20170
US

Email: housley@vigilsec.com
Problem Statement and Requirements for Header Protection
draft-ietf-lamps-header-protection-requirements-00

Abstract

Privacy and security issues with email header protection in S/MIME have been identified for some time. However, the desire to fix these issue has been expressed in the IETF LAMPS Working Group only recently. The existing S/MIME specification is likely to be updated regarding header protection.

Several LAMPS WG participants expressed the opinion that whatever mechanism will be chosen, it should not be limited to S/MIME, but also applicable to PGP/MIME.

This document describes the problem statement, generic use cases, and requirements. Additionally it drafts possible solutions to address the challenge. Finally some best practices are collected.

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 January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
# Table of Contents

1. Introduction ............................................. 3
   1.1. Requirements Language ................................. 4
   1.2. Terms ................................................. 4
2. Problem Statement ....................................... 4
3. Use Cases ................................................ 5
   3.1. Interactions ........................................ 5
   3.2. Protection Levels .................................... 6
4. Requirements .............................................. 6
   4.1. General Requirements ................................. 6
   4.1.1. Sending Side ...................................... 7
   4.1.2. Receiving Side .................................... 7
   4.2. Additional Requirements for Backward-Compatibility With Legacy Clients Unaware of Header Protection ....... 8
         4.2.1. Sending side .................................... 8
         4.2.2. Receiving side .................................. 8
   4.3. Additional Requirements for Backward-Compatibility with Legacy Header Protection Systems (if supported) ...... 8
         4.3.1. Sending Side .................................... 9
         4.3.2. Receiving Side .................................. 9
5. Options to Achieve Header Protection ...................... 9
   5.1. Option 1: Memory Hole ................................ 9
   5.2. Option 2: Wrapping with message/rfc822 or message/global .... 9
         5.2.1. Content-Type Parameter "forwarded" ............. 10
   5.3. Option 2.1: Progressive Header Disclosure .......... 10
   5.4. Examples .............................................. 11
         5.4.1. Option 1: Memory Hole ............................ 12
         5.4.2. Option 2: Wrapping with message/rfc822 or message/global .......... 13
         5.4.3. Option 2.1 Progressive Header Disclosure ...... 14
6. Sending Side Considerations ................................ 14
   6.1. Candidate Header Fields for Header Protection ......... 14
7. Receiving Side Considerations ............................. 16
   7.1. Which Header Fields to Display to User ................ 16
   7.2. Mail User Agent Algorithm for deciding which version of a header field to display .......................... 16
8. Security Considerations ................................... 16

Melnikov & Hoeneisen    Expires January 9, 2020 [Page 2]
A range of protocols for the protection of electronic mail (email) exist, which allow to assess the authenticity and integrity of the email headers section or selected header fields from the domain-level perspective, specifically DomainKeys Identified Mail (DKIM) [RFC6376] and Sender Policy Framework (SPF) [RFC7208] and Domain-based Message Authentication, Reporting, and Conformance (DMARC) [RFC7489]. These protocols, while essential to responding to a range of attacks on email, do not offer full end-to-end protection to the headers section and are not capable of providing privacy for the information contained therein.

The need for means of Data Minimization, which includes data spareness and the hiding of all technically concealable information whenever possible, has grown in importance over the past years. A standard for end-to-end protection of the email headers section exists for S/MIME since version 3.1. (cf. [RFC8551]):

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.

No mechanism for header protection has been standardized for PGP (Pretty Good Privacy) yet.

End-to-end protection for the email headers section is currently not widely implemented - neither for messages protected by means of S/MIME nor PGP. At least two variants of header protection are known to be implemented.
This document describes the problem statement, generic use cases (Section 3) and requirements for header protection (Section 4). Additionally it drafts possible solutions to address the challenge. However, the final solution will be determined by the IETF LAMPS WG. Finally, some best practices are collected.

[[ TODO: enhance this section ]]

1.1. Requirements Language

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

1.2. Terms

The following terms are defined for the scope of this document:

- Header Field: cf. [RFC5322]
- Header Section: cf. [RFC5322]
- Signed-only message: a multipart/signed or application/pkcs7-mime containing SignedData message which doesn’t contain any encrypted layer. I.e. this is a message which is not encrypted and not encrypted + signed.
- Man-in-the-middle (MITM) attack: cf. [RFC4949], which states: "A form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data to masquerade as one or more of the entities involved in a communication association."

2. Problem Statement

The LAMPS charter contains the following Work Item:

Update the specification for the cryptographic protection of email headers - both for signatures and encryption - to improve the implementation situation with respect to privacy, security, usability and interoperability in cryptographically-protected electronic mail. Most current implementations of cryptographically-protected electronic mail protect only the body of the message, which leaves significant room for attacks against otherwise-protected messages.

[[ TODO: enhance this section ]]
3. Use Cases

In the following, we show the generic use cases that need to be addressed independently of whether S/MIME, PGP/MIME or any other technology is used for which Header Protection (HP) is to be applied to.

3.1. Interactions

The main interaction case for Header Protection (HP) is:

1) Both peers (sending and receiving side) fully support HP

For backward compatibility of legacy clients – unaware of any HP – the following intermediate interactions need to be considered as well:

2) The sending side fully supports HP, while the receiving side does not support any HP

3) The sending side does not support any HP, while the receiving side fully supports HP (trivial case)

4) Neither the sending side nor the receiving side supports any HP (trivial case)

The following intermediate use cases may need to be considered as well for backward compatibility with legacy HP systems, such as S/MIME since version 3.1 (cf. [RFC8551]), in the following designated as legacy HP:

5) The sending side fully supports HP, while the receiving side supports legacy HP only

6) The sending side supports legacy HP only, while the receiving side fully supports HP

7) Both peers (sending and receiving side) support legacy HP only

8) The sending side supports legacy HP only, while the receiving side does not support any HP

9) The sending side does not support any HP, while the receiving side supports legacy HP only (trivial case)
Note: It is to be decided whether to ensure legacy HP systems do not conflict with any new solution for HP at all or whether (and to which degree) backward compatibility to legacy HP systems shall be maintained.

[[ TODO: Decide in which form legacy HP requirements should remain in this document. ]]

3.2. Protection Levels

The following protection levels need to be considered:

a) signature and encryption

b) signature only

c) encryption only

[[ TODO: verify whether relevant ]]•

4. Requirements

In the following a list of requirements that need to be addressed independently of whether S/MIME, PGP/MIME or any other technology is used to apply HP to.

4.1. General Requirements

This subsection is listing the requirements to address use case 1) (cf. Section 3.1).
G1: Define the format for HP for all protection levels MIME structure, Content-Type (including all parameters, such as "charset" and "name"), Content-Disposition (including all parameters, such as "filename"), and Content-Transfer-Encoding.

G2: To foster wide implementation of the new solution, it shall be easily implementable. Unless needed for maximizing protection and privacy, existing implementations shall not require substantial changes in the existing code base. In particular also MIME libraries widely used shall not need to be changed to comply with the new mechanism for HP.

G3: There SHOULD be only one format that covers all Protection Levels (cf. (protection-levels)))

[[ TODO: Should this one remain in the document? If yes, consider improve / rewrite sentence ]]

G4: Ensure that man-in-the-middle attack (MITM) cf. {{RFC4949}}, in particular downgrade attacks, are mitigated as good as possible.

4.1.1. Sending Side

GS1: Determine which Header Fields (HFs) should or must be protected at least for signed only email.

GS2: Determine which HFs should or must be sent in clear of an encrypted email.

GS3: Determine which HF should not or must not be included in the visible header (for transport) of an encrypted email, with the default being that whatever is not needed from GS2 is not put into the unencrypted transport headers, thus fulfilling data minimization requirements (including data spareness and hiding of all information that technically can be hidden).

GS4: Determine which HF to not to include to any HP part (e.g. Bcc).

4.1.2. Receiving Side
GR1: Determine how HF should be displayed to the user in case of conflicting information between the protected and unprotected headers.

GR2: Ensure that man-in-the-middle attack (MITM) cf. ((RFC4949)), in particular downgrade attacks, can be detected.

4.2. Additional Requirements for Backward-Compatibility With Legacy Clients Unaware of Header Protection

This sub-section addresses the use cases 2) - 4) (cf. Section 3.1)

B1: Depending on the solution, define a means to distinguish between forwarded messages and encapsulated messages using new HP mechanism.

4.2.1. Sending side

BS1: Define how full HP support can be indicated to outgoing messages.

BS2: Define how full HP support of the receiver can be detected or guessed.

BS3: Ensure a HP unaware receiving side easily can display the "Subject" HF to the user.

4.2.2. Receiving side

BR1: Define how full HP support can be detected in incoming messages.

4.3. Additional Requirements for Backward-Compatibility with Legacy Header Protection Systems (if supported)

This sub-section addresses the use cases 5) - 9) (cf. Section 3.1).

LS1: Depending on the solution, define a means to distinguish between forwarded messages, legacy encapsulated messages, and encapsulated messages using new HP mechanism.

LS2: The solution should be backward compatible to existing solutions and aim to minimize the implementation effort to include support for existing solutions.
4.3.1. Sending Side

LSS1: Determine how legacy HP support can be indicated to outgoing messages.

LSS2: Determine how legacy HP support of the receiver can be detected or guessed.

4.3.2. Receiving Side

LSR1: Determine how legacy HP support can be detected in incoming messages.

5. Options to Achieve Header Protection

In the following a set of Options to achieve Email Header Protection. It is expected that the IETF LAMPS WG chooses an option to update [RFC8551] wrt. Header Protection.

5.1. Option 1: Memory Hole

Memory Hole approach works by copying the normal message header fields into the MIME header section of the top level protected body part. Since the MIME body part header section is itself covered by the protection mechanisms (signing and/or encryption) it shares the protections of the message body.

[[ TODO: add more information on memory hole ]]

5.2. Option 2: Wrapping with message/rfc822 or message/global

Wrapping with message/rfc822 (or message/global) works by copying the normal message header fields into the MIME header section of the top level protect body part

[[ TODO: consider rephrasing, as not only the header fields is copied, but also the content.]]

and then prepending them with "Content-Type: message/rfc822; forwarded=no\r\n" or "Content-Type: message/global; forwarded=no\r\n", where \r\n is US-ASCII CR followed by US-ASCII LF. Since the MIME body part header section is itself covered by the protection mechanisms (signing and/or encryption) it shares the protections of the message body.
5.2.1. Content-Type Parameter "forwarded"

This section outlines how the new "forwarded" Content-Type header field parameter could be defined (probably in a separate document) and how header section wrapping works:

This document defines a new Content-Type header field parameter [RFC2045] with name "forwarded". The parameter value is case-insensitive and can be either "yes" or "no". (The default value being "yes"). The parameter is only meaningful with media type "message/rfc822" and "message/global" [RFC6532] when used within S/MIME signed or encrypted body parts. The value "yes" means that the message nested inside "message/rfc822" ("message/global") is a forwarded message and not a construct created solely to protect the inner header section.

Instructions in [RFC8551] describing how to protect the Email message header section [RFC5322], by wrapping the message inside a message/rfc822 container [RFC2045] are thus updated to read:

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 section along with the unprotected "outer" header section.

When an S/MIME message is received, if the top-level protected MIME entity has a Content-Type of message/rfc822 or message/global without the "forwarded" parameter or with the "forwarded" parameter set to "no", 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 section merging issues as previously discussed.

5.3. Option 2.1: Progressive Header Disclosure

This option is similar to Option 2 (cf. Section 5.2). It also makes use the Content-Type parameter "forwarded" (cf. Section 5.2.1).

pEp for email [I-D.marques-pep-email] defines a fixed MIME structure for its innermost message structure. Security comes just next after privacy in pEp, for which reason the application of signatures without encryption to messages in transit is not considered purposeful. pEp for email, either expects to transfer messages in cleartext without signature or encryption, or transfer them encrypted...
and with enclosed signature and necessary public keys so that replies can be immediately upgraded to encrypted messages.

The pEp message format is equivalent to the S/MIME standard in ensuring header protection, in that the whole message is protected instead, by wrapping it and providing cryptographic services to the whole original message. However, for the purpose of allowing the insertion of public keys, the root entity of the protected message is thus nested once more into an additional multipart/mixed MIME entity. The current pEp proposal is for PGP/MIME, while an extension to S/MIME is also on the roadmap.

pEp has also implemented the above (in Section 5.2.1) described Content-Type parameter "forwarded" to distinguish between encapsulated and forwarded emails.

More information on progressive header disclosure can be found in [I-D.luck-lamps-pep-header-protection].

5.4. Examples

Examples in subsequent sections assume that an email client is trying to protect (sign) the following initial message:
This is an important message that I don’t want to be modified.

Without message header protection the corresponding signed message might look like this. (Lines prepended by "O: " are the outer header.)

O: Date: Mon, 25 Sep 2017 17:31:42 +0100 (GMT Daylight Time)
O: Message-ID: <e4a483cb-1dfb-481d-903b-298c92c21f5e@matt.example.net>
O: Subject: Meeting at my place
O: From: "Alexey Melnikov" <alexey.melnikov@example.net>
O: MIME-Version: 1.0
O: content-type: multipart/signed; charset=us-ascii; micalg=sha1;
O: protocol="application/pkcs7-signature";
O: boundary=.cbe16d2a-e1a3-4220-b821-38348fc97237

This is a multipart message in MIME format.
--.cbe16d2a-e1a3-4220-b821-38348fc97237
Content-Type: text/plain; charset=us-ascii

This is an important message that I don’t want to be modified.

--.cbe16d2a-e1a3-4220-b821-38348fc97237
Content-Transfer-Encoding: base64
content-type: application/pkcs7-signature

[[base-64 encoded signature]]

--.cbe16d2a-e1a3-4220-b821-38348fc97237--

5.4.1. Option 1: Memory Hole

The following example demonstrates how header section and payload of a protect body part might look like. For example, this will be the first body part of a multipart/signed message or the signed and/or encrypted payload of the application/pkcs7-mime body part. Lines
5.4.2. Option 2: Wrapping with message/rfc822 or message/global

The following example demonstrates how header section and payload of a protect body part might look like. For example, this will be the first body part of a multipart/signed message or the signed and/or encrypted payload of the application/pkcs7-mime body part. Lines prepended by "O: " are the outer header section. Lines prepended by "I: " are the inner header section. Lines prepended by "W: " are the wrapper.
This is a multipart message in MIME format.

This is an important message that I don’t want to be modified.

5.4.3. Option 2.1 Progressive Header Disclosure

This looks similar as in option 2. Specific examples can be found in [I-D.luck-lamps-pep-header-protection].

6. Sending Side Considerations

6.1. Candidate Header Fields for Header Protection

[M Melanie & Hoeneisen Expires January 9, 2020 [Page 14]
For a signed-only message, it is RECOMMENDED that all "outer" header fields are identical to the "inner" protected header fields. This would mean that all header fields are signed. In this case, the "outer" header fields simply match the protected header fields. And in the case that the "outer" header fields differ, they can simply be replaced with their protected versions when displayed to the user.

[[ TODO: Decide whether "Bcc" header field should be excluded. Also verify whether this requirement applies generally or just for specific implementations. ]]

When generating encrypted or encrypted+signed S/MIME messages which protect header fields:

1. If a header field is being encrypted because it is sensitive, its true value MUST NOT be included in the outer header. If the header field is mandatory according to [RFC5322], a stub value (or a value indicating that the outer value is not to be used) is to be included in the outer header section.

2. The outer header section SHOULD be minimal in order to avoid disclosure of confidential information. It is recommended that the outer header section only contains "Date" (set to the same value as in the inner header field, or, if the Date value is also sensitive, to Monday 9am of the same week), possibly "Subject" and "To"/"Bcc" header fields. ("From", "Date", and at least one destination header field is mandatory as per [RFC5322].) In particular, Keywords, In-Reply-To and References header fields SHOULD NOT be included in the outer header; "To" and "Cc" header fields should be omitted and replaced with "Bcc: undisclosed-recipients;".

But note that having key header fields duplicated in the outer header is convenient for many message stores (e.g. IMAP) and clients that can’t decode S/MIME encrypted messages. In particular, Subject/To/Cc/Bcc/Date header field values are returned in IMAP ENVELOPE FETCH data item [RFC3501], which is frequently used by IMAP clients in order to avoid parsing message header.

3. The "Subject" header field value of the outer header section SHOULD either be identical to the inner "Subject" header field value, or contain a clear indication that the outer value is not to be used for display (the inner header field value would contain the true value).
Note that recommendations listed above typically only apply to non MIME header fields (header fields with names not starting with "Content-") prefix), but there are exception, e.g. Content-Language.

Note that the above recommendations can also negatively affect anti-spam processing.

7. Receiving Side Considerations

7.1. Which Header Fields to Display to User

When displaying S/MIME messages which protect header fields (whether they are signed-only, encrypted or encrypted+signed):

1. The outer header fields might be tampered with, so a receiving client SHOULD ignore them, unless they are protected in some other way(_). If a header field is present in the inner header, only the inner header field value MUST be displayed (and the corresponding outer value must be ignored). If a particular header field is only present in the outer header, it MAY be ignored (not displayed) or it MAY be displayed with a clear indicator that it is not trustworthy(_).

(*) - this only applies if the header field is not protected is some other way, for example with a DKIM signature that validates and is trusted.

7.2. Mail User Agent Algorithm for deciding which version of a header field to display

[[ TODO: describe how to recurse to find the innermost protected root body part, extract header fields from it and propagate them to the top level. This should also work for triple-wrapped messages.]]

8. Security Considerations

This document talks about UI considerations, including security considerations, when processing messages protecting header fields. One of the goals of this document is to specify UI for displaying such messages which is less confusing/misleading and thus more secure.

The document is not defining new protocol, so it doesn’t create any new security concerns not already covered by S/MIME [RFC8551], MIME [RFC2045] and Email [RFC5322] in general.
9. Privacy Considerations

[[ TODO ]]

10. IANA Considerations

This document requests no action from IANA.

[[ RFC Editor: This section may be removed before publication. ]]

11. Acknowledgments

The authors would like to thank the following people who have provided helpful comments and suggestions for this document: David Wilson, Steve Kille, Wei Chuang, and Robert Williams

Essential parts of [I-D.luck-lamps-pep-header-protection] have been merged into this document. Special thanks to its author Claudio Luck. For further Acknowledgments, please refer to Acknowledgments section of [I-D.luck-lamps-pep-header-protection].

David Wilson came up with the idea of defining a new Content-Type header field parameter to distinguish forwarded messages from inner header field protection constructs.

12. References

12.1. Normative References


12.2. Informative References

[I-D.luck-lamps-pep-header-protection]

[I-D.marques-pep-email]


Appendix A. Document Changelog

[ RFC Editor: This section is to be removed before publication ]

- draft-ietf-lamps-header-protection-requirements-00

  * Initial version
Appendix B. Open Issues

- Enhance Introduction and Problem Statement sections
- Decide in which form legacy HP requirements should remain in this document
- Signed-only protection needs further study
  * pEp only does header protection by applying both signing and encryption. Technically it is also possible to sign, but not encrypt the protected messages. This needs further study. Feedback from IETF-104: Probably no need to specify it, but need to document the case.
- Should requirement G3 remain? If you consider improve / rewrite it.
- Add more text on Memory Hole
- Rephrase Section 5.2
- Add example to Section 5.4.3
- Resolve question regarding Bcc in Section 6.1
- Rewrite Section 6.1
- Write Section 7.2
- Correct terminology for Header(s) and Header Fields throughout the document (editorial).
  * Header: Whole Header Section of the message
  * Header Field: Part / single Line inside a Header (Section)
Abstract

Digital signatures are used to sign messages, X.509 certificates and CRLs. This document updates the "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile" (RFC3279) and describes the conventions for using the SHAKE function family in Internet X.509 certificates and revocation lists as one-way hash functions with the RSA Probabilistic signature and ECDSA signature algorithms. The conventions for the associated subject public keys are also described.

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 January 22, 2020.

Copyright Notice

Copyright (c) 2019 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. Change Log ......................................................... 2
2. Introduction ...................................................... 5
3. Terminology ....................................................... 5
4. Identifiers ......................................................... 5
5. Use in PKIX ........................................................ 6
   5.1. Signatures .................................................... 6
   5.1.1. RSASSA-PSS Signatures .................................. 7
   5.1.2. ECDSA Signatures ......................................... 8
   5.2. Public Keys .................................................. 9
6. IANA Considerations ............................................... 9
7. Security Considerations .......................................... 10
8. Acknowledgements ................................................ 10
9. References ........................................................ 11
   9.1. Normative References ....................................... 11
   9.2. Informative References ..................................... 12
Appendix A. ASN.1 module ............................................ 13
Authors’ Addresses ................................................ 17

1. Change Log

[ EDNOTE: Remove this section before publication. ]

o draft-ietf-lamps-pkix-shake-15:
   * Minor editorial nits.

o draft-ietf-lamps-pkix-shake-14:
   * Fixing error with incorrect preimage resistance bits for SHA128 and SHA256.

o draft-ietf-lamps-pkix-shake-13:
   * Addressing one applicable comment from Dan M. about sec levels while in secdir review of draft-ietf-lamps-cms-shakes.
   * Addressing comment from Scott B.’s opsdir review about references in the abstract.

o draft-ietf-lamps-pkix-shake-12:
* Nits identified by Roman, Eric V. Ben K., Barry L. in ballot position review.

- draft-ietf-lamps-pkix-shake-11:
  * Nits identified by Roman in AD Review.

- draft-ietf-lamps-pkix-shake-10:
  * Updated IANA considerations section to request for OID assignments.

- draft-ietf-lamps-pkix-shake-09:
  * Fixed minor text nits.
  * Added text name allocation for SHAKEs in IANA considerations.
  * Updates in Sec Considerations section.

- draft-ietf-lamps-pkix-shake-08:
  * Small nits from Russ while in WGLC.

- draft-ietf-lamps-pkix-shake-07:
  * Incorporated Eric’s suggestion from WGLC.

- draft-ietf-lamps-pkix-shake-06:
  * Added informative references.
  * Updated ASN.1 so it compiles.
  * Updated IANA considerations.

- draft-ietf-lamps-pkix-shake-05:
  * Added RFC8174 reference and text.
  * Explicitly explained why RSASSA-PSS-params are omitted in section 5.1.1.
  * Simplified Public Keys section by removing redundant info from RFCs.

- draft-ietf-lamps-pkix-shake-04:
* Removed paragraph suggesting KMAC to be used in generating k in Deterministic ECDSA. That should be RFC6979-bis.

* Removed paragraph from Security Considerations that talks about randomness of k because we are using deterministic ECDSA.

* Various ASN.1 fixes.

* Text fixes.

  o draft-ietf-lamps-pkix-shake-03:
    * Updates based on suggestions and clarifications by Jim.
    * Added ASN.1.

  o draft-ietf-lamps-pkix-shake-02:
    * Significant reorganization of the sections to simplify the introduction, the new OIDs and their use in PKIX.
    * Added new OIDs for RSASSA-PSS that hardcode hash, salt and MGF, according the WG consensus.
    * Updated Public Key section to use the new RSASSA-PSS OIDs and clarify the algorithm identifier usage.
    * Removed the no longer used SHAKE OIDs from section 3.1.
    * Consolidated subsection for message digest algorithms.
    * Text fixes.

  o draft-ietf-lamps-pkix-shake-01:
    * Changed titles and section names.
    * Removed DSA after WG discussions.
    * Updated shake OID names and parameters, added MGF1 section.
    * Updated RSASSA-PSS section.
    * Added Public key algorithm OIDs.
    * Populated Introduction and IANA sections.

  o draft-ietf-lamps-pkix-shake-00:
2. Introduction

[RFC3279] defines cryptographic algorithm identifiers for the Internet X.509 Certificate and Certificate Revocation Lists (CRL) profile [RFC5280]. This document updates RFC3279 and defines identifiers for several cryptographic algorithms that use variable length output SHAKE functions introduced in [SHA3] which can be used with.

In the SHA-3 family, two extendable-output functions (SHAKEs), SHAKE128 and SHAKE256, are defined. Four other hash function instances, SHA3-224, SHA3-256, SHA3-384, and SHA3-512, are also defined but are out of scope for this document. A SHAKE is a variable length hash function defined as SHAKE(M, d) where the output is a d-bits-long digest of message M. The corresponding collision and second-preimage-resistance strengths for SHAKE128 are \( \min(d/2,128) \) and \( \min(d,128) \) bits, respectively (Appendix A.1 [SHA3]). And the corresponding collision and second-preimage-resistance strengths for SHAKE256 are \( \min(d/2,256) \) and \( \min(d,256) \) bits, respectively.

A SHAKE can be used as the message digest function (to hash the message to be signed) in RSASSA-PSS [RFC8017] and ECDSA [X9.62] and as the hash in the mask generation function (MGF) in RSASSA-PSS.

3. Terminology

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.

4. Identifiers

This section defines four new object identifiers (OIDs), for RSASSA-PSS and ECDSA with each of SHAKE128 and SHAKE256. The same algorithm identifiers can be used for identifying a public key in RSASSA-PSS.

The new identifiers for RSASSA-PSS signatures using SHAKEs are below.
id-RSASSA-PSS-SHAKE128  OBJECT IDENTIFIER  ::=  { iso(1)
    identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) algorithms(6)
    TBD1 }

id-RSASSA-PSS-SHAKE256  OBJECT IDENTIFIER  ::=  { iso(1)
    identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) algorithms(6)
    TBD2 }

The new algorithm identifiers of ECDSA signatures using SHAKEs are below.

id-ecdsa-with-shake128 OBJECT IDENTIFIER  ::=  { iso(1)
    identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) algorithms(6)
    TBD3 }

id-ecdsa-with-shake256 OBJECT IDENTIFIER  ::=  { iso(1)
    identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) algorithms(6)
    TBD4 }

The parameters for the four identifiers above MUST be absent. That is, the identifier SHALL be a SEQUENCE of one component, the OID.

Section 5.1.1 and Section 5.1.2 specify the required output length for each use of SHAKE128 or SHAKE256 in RSASSA-PSS and ECDSA. In summary, when hashing messages to be signed, output lengths of SHAKE128 and SHAKE256 are 256 and 512 bits respectively. When the SHAKEs are used as mask generation functions RSASSA-PSS, their output length is \((8 \times \text{ceil}(\frac{(n-1)}{8}) - 264)\) or \((8 \times \text{ceil}(\frac{(n-1)}{8}) - 520)\) bits, respectively, where \(n\) is the RSA modulus size in bits.

5. Use in PKIX

5.1. Signatures

Signatures are used in a number of different ASN.1 structures. As shown in the ASN.1 representation from [RFC5280] below, in an X.509 certificate, a signature is encoded with an algorithm identifier in the signatureAlgorithm attribute and a signatureValue attribute that contains the actual signature.
Certificate ::= SEQUENCE {
  tbsCertificate          TBSCertificate,
  signatureAlgorithm      AlgorithmIdentifier,
  signatureValue          BIT STRING
}

The identifiers defined in Section 4 can be used as the
AlgorithmIdentifier in the signatureAlgorithm field in the sequence
Certificate and the signature field in the sequence TBSCertificate in
X.509 [RFC5280]. The parameters of these signature algorithms are
absent as explained in Section 4.

Conforming CA implementations MUST specify the algorithms explicitly
by using the OIDs specified in Section 4 when encoding RSASSA-PSS or
ECDSA with SHAKE signatures in certificates and CRLs. Conforming
client implementations that process certificates and CRLs using
RSASSA-PSS or ECDSA with SHAKE MUST recognize the corresponding OIDs.
Encoding rules for RSASSA-PSS and ECDSA signature values are
specified in [RFC4055] and [RFC5480], respectively.

When using RSASSA-PSS or ECDSA with SHAKEs, the RSA modulus and ECDSA
curve order SHOULD be chosen in line with the SHAKE output length.
Refer to Section 7 for more details.

5.1.1. RSASSA-PSS Signatures

The RSASSA-PSS algorithm is defined in [RFC8017]. When id-RSASSA-
PSS-SHAKE128 or id-RSASSA-PSS-SHAKE256 specified in Section 4 is
used, the encoding MUST omit the parameters field. That is, the
AlgorithmIdentifier SHALL be a SEQUENCE of one component, id-RSASSA-
PSS-SHAKE128 or id-RSASSA-PSS-SHAKE256. [RFC4055] defines RSASSA-
PSS-params that are used to define the algorithms and inputs to the
algorithm. This specification does not use parameters because the
hash, mask generation algorithm, trailer and salt are embedded in the
OID definition.

The hash algorithm to hash a message being signed and the hash
algorithm used as the mask generation function in RSASSA-PSS MUST be
the same: both SHAKE128 or both SHAKE256. The output length of the
hash algorithm which hashes the message SHALL be 32 (for SHAKE128) or
64 bytes (for SHAKE256).

The mask generation function takes an octet string of variable length
and a desired output length as input, and outputs an octet string of
the desired length. In RSASSA-PSS with SHAKEs, the SHAKEs MUST be
used natively as the MGF function, instead of the MGF1 algorithm that
uses the hash function in multiple iterations as specified in
Section B.2.1 of [RFC8017]. In other words, the MGF is defined as
the SHAKE128 or SHAKE256 output of the mgfSeed for id-RSASSA-PSS-
SHAKE128 and id-RSASSA-PSS-SHAKE256, respectively. The mgfSeed is
the seed from which mask is generated, an octet string [RFC8017]. As
explained in Step 9 of section 9.1.1 of [RFC8017], the output length
of the MGF is emLen - hLen - 1 bytes. emLen is the maximum message
length ceil((n-1)/8), where n is the RSA modulus in bits. hLen is 32
and 64-bytes for id-RSASSA-PSS-SHAKE128 and id-RSASSA-PSS-SHAKE256,
respectively. Thus when SHAKE is used as the MGF, the SHAKE output
length maskLen is (8*emLen - 264) or (8*emLen - 520) bits,
respectively. For example, when RSA modulus n is 2048, the output
length of SHAKE128 or SHAKE256 as the MGF will be 1784 or 1528-bits
when id-RSASSA-PSS-SHAKE128 or id-RSASSA-PSS-SHAKE256 is used,
respectively.

The RSASSA-PSS saltLength MUST be 32 bytes for id-RSASSA-PSS-SHAKE128
or 64 bytes for id-RSASSA-PSS-SHAKE256. Finally, the trailerField
MUST be 1, which represents the trailer field with hexadecimal value
0xBC [RFC8017].

5.1.2. ECDSA Signatures

The Elliptic Curve Digital Signature Algorithm (ECDSA) is defined in
[X9.62]. When the id-ecdsa-with-shake128 or id-ecdsa-with-shake256
(specified in Section 4) algorithm identifier appears, the respective
SHAKE function (SHAKE128 or SHAKE256) is used as the hash. The
encoding MUST omit the parameters field. That is, the
AlgorithmIdentifier SHALL be a SEQUENCE of one component, the OID id-
ecdsa-with-shake128 or id-ecdsa-with-shake256.

For simplicity and compliance with the ECDSA standard specification,
the output length of the hash function must be explicitly determined.
The output length, d, for SHAKE128 or SHAKE256 used in ECDSA MUST be
256 or 512 bits, respectively.

Conforming CA implementations that generate ECDSA with SHAKE
signatures in certificates or CRLs SHOULD generate such signatures
with a deterministically generated, non-random k in accordance with
all the requirements specified in [RFC6979]. They MAY also generate
such signatures in accordance with all other recommendations in
[X9.62] or [SEC1] if they have a stated policy that requires
conformance to those standards. Those standards have not specified
SHAKE128 and SHAKE256 as hash algorithm options. However, SHAKE128
and SHAKE256 with output length being 32 and 64 octets, respectively,
can be used instead of 256 and 512-bit output hash algorithms such as
SHA256 and SHA512.
5.2. Public Keys

Certificates conforming to [RFC5280] can convey a public key for any public key algorithm. The certificate indicates the public key algorithm through an algorithm identifier. This algorithm identifier is an OID and optionally associated parameters. The conventions and encoding for RSASSA-PSS and ECDSA public keys algorithm identifiers are as specified in Section 2.3.1 and 2.3.5 of [RFC3279], Section 3.1 of [RFC4055] and Section 2.1 of [RFC5480].

Traditionally, the rsaEncryption object identifier is used to identify RSA public keys. The rsaEncryption object identifier continues to identify the subject public key when the RSA private key owner does not wish to limit the use of the public key exclusively to RSASSA-PSS with SHAKEs. When the RSA private key owner wishes to limit the use of the public key exclusively to RSASSA-PSS with SHAKEs, the AlgorithmIdentifiers for RSASSA-PSS defined in Section 4 SHOULD be used as the algorithm field in the SubjectPublicKeyInfo sequence [RFC5280]. Conforming client implementations that process RSASSA-PSS with SHAKE public keys when processing certificates and CRLs MUST recognize the corresponding OIDs.

Conforming CA implementations MUST specify the X.509 public key algorithm explicitly by using the OIDs specified in Section 4 when encoding ECDSA with SHAKE public keys in certificates and CRLs. Conforming client implementations that process ECDSA with SHAKE public keys when processing certificates and CRLs MUST recognize the corresponding OIDs.

The identifier parameters, as explained in Section 4, MUST be absent.

6. IANA Considerations

One object identifier for the ASN.1 module in Appendix A is requested for the SMI Security for PKIX Module Identifiers (1.3.6.1.5.5.7.0) registry:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Description</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD</td>
<td>id-mod-pkix1-shakes-2019</td>
<td>[EDNOTE: THIS RFC]</td>
</tr>
</tbody>
</table>

IANA is requested to update the SMI Security for PKIX Algorithms [SMI-PKIX] (1.3.6.1.5.5.7.6) registry with four additional entries:
IANA is also requested to update the Hash Function Textual Names Registry [Hash-Texts] with two additional entries for SHAKE128 and SHAKE256:

<table>
<thead>
<tr>
<th>Hash Function Name</th>
<th>OID</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>shake128</td>
<td>2.16.840.1.101.3.4.2.11</td>
<td>[EDNOTE: THIS RFC]</td>
</tr>
<tr>
<td>shake256</td>
<td>2.16.840.1.101.3.4.2.12</td>
<td>[EDNOTE: THIS RFC]</td>
</tr>
</tbody>
</table>

7. Security Considerations

This document updates [RFC3279]. The security considerations section of that document applies to this specification as well.

NIST has defined appropriate use of the hash functions in terms of the algorithm strengths and expected time frames for secure use in Special Publications (SPs) [SP800-78-4] and [SP800-107]. These documents can be used as guides to choose appropriate key sizes for various security scenarios.

SHAKE128 with output length of 256-bits offers 128-bits of collision and preimage resistance. Thus, SHAKE128 OIDs in this specification are RECOMMENDED with 2048 (112-bit security) or 3072-bit (128-bit security) RSA modulus or curves with group order of 256-bits (128-bit security). SHAKE256 with 512-bits output length offers 256-bits of collision and preimage resistance. Thus, the SHAKE256 OIDs in this specification are RECOMMENDED with 4096-bit RSA modulus or higher or curves with group order of at least 521-bits (256-bit security). Note that we recommended 4096-bit RSA because we would need 15360-bit modulus for 256-bits of security which is impractical for today’s technology.

8. Acknowledgements

We would like to thank Sean Turner, Jim Schaad and Eric Rescorla for their valuable contributions to this document.
The authors would like to thank Russ Housley for his guidance and very valuable contributions with the ASN.1 module.

9. References

9.1. Normative References


9.2. Informative References

[Hash-Texts]
IANA, "Hash Function Textual Names", July 2017,
<https://www.iana.org/assignments/hash-function-text-names/hash-function-text-names.xhtml>.

[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,

[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

[SEC1] Standards for Efficient Cryptography Group, "SEC 1:
Elliptic Curve Cryptography", May 2009,

[SMI-PKIX]
IANA, "SMI Security for PKIX Algorithms", March 2019,
<https://www.iana.org/assignments/smi-numbers/
smi-numbers.xhtml#smi-numbers-1.3.6.1.5.5.7.6>.

[SP800-107]
National Institute of Standards and Technology (NIST),
"SP800-107: Recommendation for Applications Using Approved
Hash Algorithms", May 2014,
<https://csrc.nist.gov/csrc/media/publications/sp/800-107/

[SP800-78-4]
National Institute of Standards and Technology (NIST),
"SP800-78-4: Cryptographic Algorithms and Key Sizes for
Personal Identity Verification", May 2014,
<https://csrc.nist.gov/csrc/media/publications/sp/800-78-4/
final/documents/sp800_78-4_revised_draft.pdf>.
Appendix A.  ASN.1 module

This appendix includes the ASN.1 module for SHAKEs in X.509. This module does not come from any existing RFC.

PKIXAlgsForSHAKE-2019 { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-pkix1-shakes-2019(TBD) }

DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS ALL;

IMPORTS

-- FROM [RFC5912]

PUBLIC-KEY, SIGNATURE-ALGORITHM, DIGEST-ALGORITHM, SMIME-CAPS FROM AlgorithmInformation-2009
    { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-algorithmInformation-02(58) }

-- FROM [RFC5912]

RSAPublicKey, rsaEncryption, pk-rsa, pk-ec, CURVE, id-ecPublicKey, ECPoint, ECParameters, ECDSA-Sig-Value FROM PKIXAlgs-2009
    { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-pkix1-algorithms2008-02(56) }

;

-- Message Digest Algorithms (mda-)

DigestAlgorithms DIGEST-ALGORITHM ::= {
    -- This expands DigestAlgorithms from [RFC5912]
    mda-shake128 |
    mda-shake256,
    ...
}
--
-- One-Way Hash Functions
--

-- SHAKE128
mda-shake128 DIGEST-ALGORITHM ::= {
    IDENTIFIER id-shake128  -- with output length 32 bytes.
}

id-shake128 OBJECT IDENTIFIER ::= {
joint-iso-itu-t(2) country(16)
    us(840) organization(1) gov(101)
    csor(3) nistAlgorithm(4)
    hashAlgs(2) 11
}

-- SHAKE256
mda-shake256 DIGEST-ALGORITHM ::= {
    IDENTIFIER id-shake256  -- with output length 64 bytes.
}

id-shake256 OBJECT IDENTIFIER ::= {
joint-iso-itu-t(2) country(16)
    us(840) organization(1) gov(101)
    csor(3) nistAlgorithm(4)
    hashAlgs(2) 12
}

--
-- Public Key (pk-) Algorithms
--

PublicKeys PUBLIC-KEY ::= {
    -- This expands PublicKeys from [RFC5912]
    pk-rsaSSA-PSS-SHAKE128 |
    pk-rsaSSA-PSS-SHAKE256,
    ...
}

-- The hashAlgorithm is mda-shake128
-- The maskGenAlgorithm is id-shake128
-- Mask Gen Algorithm is SHAKE128 with output length
-- (8*ceil((n-1)/8) - 264) bits, where n is the RSA
-- modulus in bits.
-- The saltLength is 32. The trailerField is 1.
pk-rsaSSA-PSS-SHAKE128 PUBLIC-KEY ::= {
    IDENTIFIER id-RSASSA-PSS-SHAKE128
    KEY RSAPublicKey
    PARAMS ARE absent
    -- Private key format not in this module --
    CERT-KEY-USAGE { nonRepudiation, digitalSignature,
        keyCertSign, cRLSign }
}

-- The hashAlgorithm is mda-shake256
The maskGenAlgorithm is id-shake256
-- Mask Gen Algorithm is SHAKE256 with output length
-- (8*ceil((n-1)/8) - 520)-bits, where n is the RSA modulus in bits.
-- The saltLength is 64. The trailerField is 1.

pk-rsaSSA-PSS-SHAKE256 PUBLIC-KEY ::= {
  IDENTIFIER id-RSASSA-PSS-SHAKE256
  KEY RSAPublicKey
  PARAMS ARE absent
  -- Private key format not in this module --
  CERT-KEY-USAGE { nonRepudiation, digitalSignature, keyCertSign, cRLSign }
}

-- Signature Algorithms (sa-)

SignatureAlgs SIGNATURE-ALGORITHM ::= {
-- This expands SignatureAlgorithms from [RFC5912]
  sa-rsassapssWithSHAKE128
  sa-rsassapssWithSHAKE256
  sa-ecdsaWithSHAKE128
  sa-ecdsaWithSHAKE256,
  ...
}

-- SMIME Capabilities (sa-)

SMimeCaps SMIME-CAPS ::= {
-- The expands SMimeCaps from [RFC5912]
  sa-rsassapssWithSHAKE128.&smimeCaps
  sa-rsassapssWithSHAKE256.&smimeCaps
  sa-ecdsaWithSHAKE128.&smimeCaps
  sa-ecdsaWithSHAKE256.&smimeCaps
  ...
}

-- RSASSA-PSS with SHAKE128
sa-rsassapssWithSHAKE128 SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-RSASSA-PSS-SHAKE128
  PARAMS ARE absent
    -- The hashAlgorithm is mda-shake128
    -- The maskGenAlgorithm is id-shake128
    -- Mask Gen Algorithm is SHAKE128 with output length
    -- (8*ceil((n-1)/8) - 264) bits, where n is the RSA modulus in bits.
    -- The saltLength is 32. The trailerField is 1
HASHES { mda-shake128 }
PUBLIC-KEYS { pk-rsa | pk-rsaSSA-PSS-SHAKE128 }
SMIME-CAPS { IDENTIFIED BY id-RSASSA-PSS-SHAKE128 }

id-RSASSA-PSS-SHAKE128 OBJECT IDENTIFIER ::= { iso(1)
  identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) algorithms(6)
  TBD1 }

-- RSASSA-PSS with SHAKE256
sa-rsassapssWithSHAKE256 SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-RSASSA-PSS-SHAKE256
  PARAMS ARE absent
  -- The hashAlgorithm is mda-shake256
  -- The maskGenAlgorithm is id-shake256
  -- Mask Gen Algorithm is SHAKE256 with output length
  -- (8*ceil((n-1)/8) - 520)-bits, where n is the
  -- RSA modulus in bits.
  -- The saltLength is 64. The trailerField is 1.
  HASHES { mda-shake256 }
  PUBLIC-KEYS { pk-rsa | pk-rsaSSA-PSS-SHAKE256 }
  SMIME-CAPS { IDENTIFIED BY id-RSASSA-PSS-SHAKE256 }
}

id-RSASSA-PSS-SHAKE256 OBJECT IDENTIFIER ::= { iso(1)
  identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) algorithms(6)
  TBD2 }

-- ECDSA with SHAKE128
sa-ecdsaWithSHAKE128 SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-ecdsa-with-shake128
  VALUE ECDSA-Sig-Value
  PARAMS ARE absent
  HASHES { mda-shake128 }
  PUBLIC-KEYS { pk-ec }
  SMIME-CAPS { IDENTIFIED BY id-ecdsa-with-shake128 }
}

id-ecdsa-with-shake128 OBJECT IDENTIFIER ::= { iso(1)
  identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) algorithms(6)
  TBD3 }

-- ECDSA with SHAKE256
sa-ecdsaWithSHAKE256 SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-ecdsa-with-shake256
  VALUE ECDSA-Sig-Value
  PARAMS ARE absent
  HASHES { mda-shake256 }

PUBLIC-KEYS { pk-ec }
SMIME-CAPS { IDENTIFIED BY id-ecdsa-with-shake256 }
}

id-ecdsa-with-shake256 OBJECT IDENTIFIER ::= { iso(1)
  identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) algorithms(6)
  TBD4 }

END

Authors’ Addresses

Panos Kampanakis
Cisco Systems
Email: pkampana@cisco.com

Quynh Dang
NIST
100 Bureau Drive, Stop 8930
Gaithersburg, MD 20899-8930
USA
Email: quynh.dang@nist.gov
Abstract

The Certification Authority Authorization (CAA) DNS Resource Record allows a DNS domain name holder to specify one or more Certification Authorities (CAs) authorized to issue certificates for that domain name. CAA Resource Records allow a public Certification Authority to implement additional controls to reduce the risk of unintended certificate mis-issue. This document defines the syntax of the CAA record and rules for processing CAA records by certificate issuers.

This document obsoletes RFC 6844.

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 December 1, 2019.

Copyright Notice

Copyright (c) 2019 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
1. Introduction

The Certification Authority Authorization (CAA) DNS Resource Record allows a DNS domain name holder to specify the Certification Authorities (CAs) authorized to issue certificates for that domain name. Publication of CAA Resource Records allows a public...
Certification Authority to implement additional controls to reduce the risk of unintended certificate mis-issue.

Like the TLSA record defined in DNS-Based Authentication of Named Entities (DANE) [RFC6698], CAA records are used as a part of a mechanism for checking PKIX [RFC6698] certificate data. The distinction between the two specifications is that CAA records specify an authorization control to be performed by a certificate issuer before issue of a certificate and TLSA records specify a verification control to be performed by a relying party after the certificate is issued.

Conformance with a published CAA record is a necessary but not sufficient condition for issuance of a certificate.

Criteria for inclusion of embedded trust anchor certificates in applications are outside the scope of this document. Typically, such criteria require the CA to publish a Certification Practices Statement (CPS) that specifies how the requirements of the Certificate Policy (CP) are achieved. It is also common for a CA to engage an independent third-party auditor to prepare an annual audit statement of its performance against its CPS.

A set of CAA records describes only current grants of authority to issue certificates for the corresponding DNS domain name. Since certificates are valid for a period of time, it is possible that a certificate that is not conformant with the CAA records currently published was conformant with the CAA records published at the time that the certificate was issued. Relying parties MUST NOT use CAA records as part of certificate validation.

CAA records MAY be used by Certificate Evaluators as a possible indicator of a security policy violation. Such use SHOULD take account of the possibility that published CAA records changed between the time a certificate was issued and the time at which the certificate was observed by the Certificate Evaluator.

2. Definitions

2.1. Requirements Language

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.
2.2. Defined Terms

The following terms are used in this document:

Certificate: An X.509 Certificate, as specified in [RFC5280].

Certificate Evaluator: A party other than a Relying Party that evaluates the trustworthiness of certificates issued by Certification Authorities.

Certification Authority (CA): An Issuer that issues certificates in accordance with a specified Certificate Policy.

Certificate Policy (CP): Specifies the criteria that a Certification Authority undertakes to meet in its issue of certificates. See [RFC3647].

Certification Practices Statement (CPS): Specifies the means by which the criteria of the Certificate Policy are met. In most cases, this will be the document against which the operations of the Certification Authority are audited. See [RFC3647].

Domain Name: The label assigned to a node in the Domain Name System.

Domain Name System (DNS): The Internet naming system specified in [RFC1034] and [RFC1035].

DNS Security (DNSSEC): Extensions to the DNS that provide authentication services as specified in [RFC4033], [RFC4034], [RFC4035], [RFC5155], and revisions.

Fully-Qualified Domain Name (FQDN): A Domain Name that includes the labels of all superior nodes in the Domain Name System.

Issuer: An entity that issues certificates. See [RFC5280].

Property: The tag-value portion of a CAA Resource Record.

Property Tag: The tag portion of a CAA Resource Record.

Property Value: The value portion of a CAA Resource Record.

Resource Record (RR): A particular entry in the DNS including the owner name, class, type, time to live, and data, as defined in [RFC1034] and [RFC2181].

Resource Record Set (RRSet): A set of Resource Records of a particular owner name, class, and type. The time to live on all RRs
within an RRSet is always the same, but the data may be different among RRs in the RRSet.

Relevant Resource Record Set (Relevant RRSet): A set of CAA Resource Records resulting from applying the algorithm in Section 3 to a specific Fully-Qualified Domain Name or Wildcard Domain Name.

Relying Party: A party that makes use of an application whose operation depends on use of a certificate for making a security decision. See [RFC5280].

Wildcard Domain Name: A Domain Name consisting of a single asterisk character followed by a single full stop character ("*.") followed by a Fully-Qualified Domain Name.

3. Relevant Resource Record Set

Before issuing a certificate, a compliant CA MUST check for publication of a Relevant RRSet. If such an RRSet exists, a CA MUST NOT issue a certificate unless the CA determines that either (1) the certificate request is consistent with the applicable CAA Resource Record set or (2) an exception specified in the relevant Certificate Policy or Certification Practices Statement applies. If the Relevant RRSet for a Fully-Qualified Domain Name or Wildcard Domain Name contains no Property Tags that restrict issuance (for instance, if it contains only iodef Property Tags, or only Property Tags unrecognized by the CA), CAA does not restrict issuance.

A certificate request MAY specify more than one Fully-Qualified Domain Name and MAY specify Wildcard Domain Names. Issuers MUST verify authorization for all the Fully-Qualified Domain Names and Wildcard Domain Names specified in the request.

The search for a CAA RRSet climbs the DNS name tree from the specified label up to but not including the DNS root ‘.’ until a CAA RRSet is found.

Given a request for a specific Fully-Qualified Domain Name X, or a request for a Wildcard Domain Name *.X, the Relevant Resource Record Set RelevantCAASet(X) is determined as follows (in pseudocode):

Let CAA(X) be the RRSet returned by performing a CAA record query for the Fully-Qualified Domain Name X, according to the lookup algorithm specified in RFC 1034 section 4.3.2 (in particular chasing aliases). Let Parent(X) be the Fully-Qualified Domain Name produced by removing the leftmost label of X.
RelevantCAASet(domain):
    while domain is not ".":
        if CAA(domain) is not Empty:
            return CAA(domain)
        domain = Parent(domain)
    return Empty

For example, processing CAA for the Fully-Qualified Domain Name "X.Y.Z" where there are no CAA records at any level in the tree RelevantCAASet would have the following steps:

    CAA("X.Y.Z.") = Empty; domain = Parent("X.Y.Z.") = "Y.Z."
    CAA("Y.Z.")   = Empty; domain = Parent("Y.Z.")   = "Z."
    CAA("Z.")     = Empty; domain = Parent("Z.")     = "."
    return Empty

Processing CAA for the Fully-Qualified Domain Name "A.B.C" where there is a CAA record "issue example.com" at "B.C" would terminate early upon finding the CAA record:

    CAA("A.B.C.") = Empty; domain = Parent("A.B.C.") = "B.C."
    CAA("B.C.")   = "issue example.com"
    return "issue example.com"

4. Mechanism

4.1. Syntax

A CAA Resource Record contains a single Property consisting of a tag-value pair. A Fully-Qualified Domain Name MAY have multiple CAA RRs associated with it and a given Property Tag MAY be specified more than once across those RRs.

The RDATA section for a CAA Resource Record contains one Property. A Property consists of the following:

```
+0-1-2-3-4-5-6-7-+ 0-1-2-3-4-5-6-7-+
| Flags             | Tag Length = n |
+-------------------+----------------+
| Tag char 0        | Tag char 1     |
|                   | ...            |
| +-------------------+----------------+
| Value byte 0      | Value byte 1   |
|                   | ...            |
| +-------------------+----------------+
```

Where n is the length specified in the Tag length field and m is the remaining octets in the Value field. They are related by \( m = d - n - 2 \) where d is the length of the RDATA section.
The fields are defined as follows:

**Flags**: One octet containing the following field:

Bit 0, Issuer Critical Flag: If the value is set to ‘1’, the Property is critical. A Certification Authority MUST NOT issue certificates for any FQDN the Relevant RRSSet for that FQDN contains a CAA critical Property for an unknown or unsupported Property Tag.

Note that according to the conventions set out in [RFC1035], bit 0 is the Most Significant Bit and bit 7 is the Least Significant Bit. Thus, the Flags value 1 means that bit 7 is set while a value of 128 means that bit 0 is set according to this convention.

All other bit positions are reserved for future use.

To ensure compatibility with future extensions to CAA, DNS records compliant with this version of the CAA specification MUST clear (set to "0") all reserved flags bits. Applications that interpret CAA records MUST ignore the value of all reserved flag bits.

**Tag Length**: A single octet containing an unsigned integer specifying the tag length in octets. The tag length MUST be at least 1.

**Tag**: The Property identifier, a sequence of US-ASCII characters.


Tags submitted for registration by IANA MUST NOT contain any characters other than the (lowercase) US-ASCII characters ‘a’ through ‘z’ and the numbers 0 through 9.

**Value**: A sequence of octets representing the Property Value. Property Values are encoded as binary values and MAY employ sub-formats.

The length of the value field is specified implicitly as the remaining length of the enclosing RDATA section.

4.1.1. Canonical Presentation Format

The canonical presentation format of the CAA record is:

```
CAA <flags> <tag> <value>
```

Where:
Flags: Is an unsigned integer between 0 and 255.

Tag: Is a non-zero-length sequence of US-ASCII letters and numbers in lower case.

Value: The value field, expressed as a contiguous set of characters without interior spaces, or as a quoted string. See the `<character-string>` format specified in [RFC1035], Section 5.1, but note that the value field contains no length byte and is not limited to 255 characters.

4.2. CAA issue Property

If the issue Property Tag is present in the Relevant RRSet for a Fully-Qualified Domain Name, it is a request that Issuers

1. Perform CAA issue restriction processing for the FQDN, and
2. Grant authorization to issue certificates containing that FQDN to the holder of the issuer-domain-name or a party acting under the explicit authority of the holder of the issuer-domain-name.

The CAA issue Property Value has the following sub-syntax (specified in ABNF as per [RFC5234]).

```
issue-value = *WSP [issuer-domain-name *WSP] [";" *WSP [parameters *WSP]]
issuer-domain-name = label *("." label)
label = (ALPHA / DIGIT) *(("-" (ALPHA / DIGIT))
parameters = (parameter *WSP ";" *WSP parameters) / parameter
parameter = tag *WSP "=" *WSP value
tag = (ALPHA / DIGIT) *( *("-" (ALPHA / DIGIT))
value = *(%x21-3A / %x3C-7E)
```

For consistency with other aspects of DNS administration, FQDN values are specified in letter-digit-hyphen Label (LDH-Label) form.

The following CAA record set requests that no certificates be issued for the FQDN ‘certs.example.com’ by any Issuer other than ca1.example.net or ca2.example.org.

```
certs.example.com         CAA 0 issue "ca1.example.net"
certs.example.com         CAA 0 issue "ca2.example.org"
```

Because the presence of an issue Property Tag in the Relevant RRSet for an FQDN restricts issuance, FQDN owners can use an issue Property Tag with no issuer-domain-name to request no issuance.
For example, the following RRSet requests that no certificates be issued for the FQDN ‘nocerts.example.com’ by any Issuer.

```
nocerts.example.com       CAA 0 issue ";"
```

An issue Property Tag where the issue-value does not match the ABNF grammar MUST be treated the same as one specifying an empty issuer-domain-name. For example, the following malformed CAA RRSet forbids issuance:

```
malformed.example.com     CAA 0 issue "%%%%%"
```

CAA authorizations are additive; thus, the result of specifying both an empty issuer-domain-name and a non-empty issuer-domain-name is the same as specifying just the non-empty issuer-domain-name.

An Issuer MAY choose to specify parameters that further constrain the issue of certificates by that Issuer, for example, specifying that certificates are to be subject to specific validation policies, billed to certain accounts, or issued under specific trust anchors.

For example, if ca1.example.net has requested its customer accountable.example.com to specify their account number "230123" in each of the customer’s CAA records using the (CA-defined) "account" parameter, it would look like this:

```
accountable.example.com   CAA 0 issue "ca1.example.net; account=230123"
```

The semantics of parameters to the issue Property Tag are determined by the Issuer alone.

### 4.3. CAA issuewild Property

The issuewild Property Tag has the same syntax and semantics as the issue Property Tag except that it only grants authorization to issue certificates that specify a Wildcard Domain Name and issuewild properties take precedence over issue properties when specified. Specifically:

issuewild properties MUST be ignored when processing a request for a Fully-Qualified Domain Name that is not a Wildcard Domain Name.

If at least one issuewild Property is specified in the Relevant RRSet for a Wildcard Domain Name, all issue properties MUST be ignored when processing a request for that Wildcard Domain Name.

For example, the following RRSet requests that _only_ ca1.example.net issue certificates for "wild.example.com" or "sub.wild.example.com",
and that _only_ ca2.example.org issue certificates for "*.wild.example.com" or "*.sub.wild.example.com"). Note that this presumes there are no CAA RRs for sub.wild.example.com.

wild.example.com CAA 0 issue "ca1.example.net"
wild.example.com CAA 0 issuewild "ca2.example.org"

The following RRSet requests that _only_ ca1.example.net issue certificates for "wild2.example.com", "*.wild2.example.com" or "*.sub.wild2.example.com".

wild2.example.com CAA 0 issue "ca1.example.net"

The following RRSet requests that _only_ ca2.example.org issue certificates for "*.wild3.example.com" or "*.sub.wild3.example.com". It does not permit any Issuer to issue for "wild3.example.com" or "sub.wild3.example.com".

wild3.example.com CAA 0 issuewild "ca2.example.org"
wild3.example.com CAA 0 issue ";"

The following RRSet requests that _only_ ca2.example.org issue certificates for "*.wild3.example.com" or "*.sub.wild3.example.com". It permits any Issuer to issue for "wild3.example.com" or "sub.wild3.example.com".

wild3.example.com CAA 0 issuewild "ca2.example.org"

4.4. CAA iodef Property

The iodef Property specifies a means of reporting certificate issue requests or cases of certificate issue for domains for which the Property appears in the Relevant RRSet, when those requests or issuances violate the security policy of the Issuer or the FQDN holder.

The Incident Object Description Exchange Format (IODEF) [RFC7970] is used to present the incident report in machine-readable form.

The iodef Property Tag takes a URL as its Property Value. The URL scheme type determines the method used for reporting:

mailto: The IODEF incident report is reported as a MIME email attachment to an SMTP email that is submitted to the mail address specified. The mail message sent SHOULD contain a brief text message to alert the recipient to the nature of the attachment.
http or https: The IODEF report is submitted as a Web service request to the HTTP address specified using the protocol specified in [RFC6546].

These are the only supported URL schemes.

The following RRSet specifies that reports may be made by means of email with the IODEF data as an attachment, a Web service [RFC6546], or both:

```
report.example.com       CAA 0 issue "ca1.example.net"
report.example.com       CAA 0 iodef "mailto:security@example.com"
report.example.com       CAA 0 iodef "http://iodef.example.com/"
```

4.5. Critical Flag

The critical flag is intended to permit future versions of CAA to introduce new semantics that MUST be understood for correct processing of the record, preventing conforming CAs that do not recognize the new semantics from issuing certificates for the indicated FQDNs.

In the following example, the Property with a Property Tag of ‘tbs’ is flagged as critical. Neither the ca1.example.net CA nor any other Issuer is authorized to issue for "new.example.com" (or any other domains for which this is the Relevant RRSet) unless the Issuer has implemented the processing rules for the ‘tbs’ Property Tag.

```
new.example.com       CAA 0 issue "ca1.example.net"
new.example.com       CAA 128 tbs "Unknown"
```

5. Security Considerations

CAA records assert a security policy that the holder of an FQDN wishes to be observed by Issuers. The effectiveness of CAA records as an access control mechanism is thus dependent on observance of CAA constraints by Issuers.

The objective of the CAA record properties described in this document is to reduce the risk of certificate mis-issue rather than avoid reliance on a certificate that has been mis-issued. DANE [RFC6698] describes a mechanism for avoiding reliance on mis-issued certificates.
5.1. Use of DNS Security

Use of DNSSEC to authenticate CAA RRs is strongly RECOMMENDED but not required. An Issuer MUST NOT issue certificates if doing so would conflict with the Relevant RRSet, irrespective of whether the corresponding DNS records are signed.

DNSSEC provides a proof of non-existence for both DNS Fully-Qualified Domain Names and RRSets within FQDNs. DNSSEC verification thus enables an Issuer to determine if the answer to a CAA record query is empty because the RRSet is empty or if it is non-empty but the response has been suppressed.

Use of DNSSEC allows an Issuer to acquire and archive a proof that they were authorized to issue certificates for the FQDN. Verification of such archives may be an audit requirement to verify CAA record processing compliance. Publication of such archives may be a transparency requirement to verify CAA record processing compliance.

5.2. Non-Compliance by Certification Authority

CAA records offer CAs a cost-effective means of mitigating the risk of certificate mis-issue: the cost of implementing CAA checks is very small and the potential costs of a mis-issue event include the removal of an embedded trust anchor.

5.3. Mis-Issue by Authorized Certification Authority

Use of CAA records does not prevent mis-issue by an authorized Certification Authority, i.e., a CA that is authorized to issue certificates for the FQDN in question by CAA records.

FQDN holders SHOULD verify that the CAs they authorize to issue certificates for their FQDNs employ appropriate controls to ensure that certificates are issued only to authorized parties within their organization.

Such controls are most appropriately determined by the FQDN holder and the authorized CA(s) directly and are thus out of scope of this document.

5.4. Suppression or Spoofing of CAA Records

Suppression of the CAA record or insertion of a bogus CAA record could enable an attacker to obtain a certificate from an Issuer that was not authorized to issue for an affected FQDN.
Where possible, Issuers SHOULD perform DNSSEC validation to detect missing or modified CAA record sets.

In cases where DNSSEC is not deployed for a corresponding FQDN, an Issuer SHOULD attempt to mitigate this risk by employing appropriate DNS security controls. For example, all portions of the DNS lookup process SHOULD be performed against the authoritative name server. Data cached by third parties MUST NOT be relied on as the sole source of DNS CAA information but MAY be used to support additional anti-spoofing or anti-suppression controls.

5.5. Denial of Service

Introduction of a malformed or malicious CAA RR could in theory enable a Denial-of-Service (DoS) attack. This could happen by modification of authoritative DNS records or by spoofing inflight DNS responses.

This specific threat is not considered to add significantly to the risk of running an insecure DNS service.

An attacker could, in principle, perform a DoS attack against an Issuer by requesting a certificate with a maliciously long DNS name. In practice, the DNS protocol imposes a maximum name length and CAA processing does not exacerbate the existing need to mitigate DoS attacks to any meaningful degree.

5.6. Abuse of the Critical Flag

A Certification Authority could make use of the critical flag to trick customers into publishing records that prevent competing Certification Authorities from issuing certificates even though the customer intends to authorize multiple providers. This could happen if the customers were setting CAA records based on data provided by the CA rather than generating those records themselves.

In practice, such an attack would be of minimal effect since any competent competitor that found itself unable to issue certificates due to lack of support for a Property marked critical should investigate the cause and report the reason to the customer. The customer will thus discover that they had been deceived.

6. Deployment Considerations

A CA implementing CAA may find that they receive errors looking up CAA records. The following are some common causes of such errors, so that CAs may provide guidance to their subscribers on fixing the underlying problems.
6.1. Blocked Queries or Responses

Some middleboxes, in particular anti-DDoS appliances, may be configured to drop DNS packets of unknown types, or may start dropping such packets when they consider themselves under attack. This generally manifests as a timed-out DNS query, or a SERVFAIL at a local recursive resolver.

6.2. Rejected Queries and Malformed Responses

Some authoritative nameservers respond with REJECTED or NOTIMP when queried for a Resource Record type they do not recognize. At least one authoritative resolver produces a malformed response (with the QR bit set to 0) when queried for unknown Resource Record types. Per RFC 1034, the correct response for unknown Resource Record types is NOERROR.

6.3. Delegation to Private Nameservers

Some FQDN administrators make the contents of a subdomain unresolvable on the public Internet by delegating that subdomain to a nameserver whose IP address is private. A CA processing CAA records for such subdomains will receive SERVFAIL from its recursive resolver. The CA MAY interpret that as preventing issuance. FQDN administrators wishing to issue certificates for private FQDNs SHOULD use split-horizon DNS with a publicly available nameserver, so that CAs can receive a valid, empty CAA response for those FQDNs.

6.4. Bogus DNSSEC Responses

Queries for CAA Resource Records are different from most DNS RR types, because a signed, empty response to a query for CAA RRs is meaningfully different from a bogus response. A signed, empty response indicates that there is definitely no CAA policy set at a given label. A bogus response may mean either a misconfigured zone, or an attacker tampering with records. DNSSEC implementations may have bugs with signatures on empty responses that go unnoticed, because for more common Resource Record types like A and AAAA, the difference to an end user between empty and bogus is irrelevant; they both mean a site is unavailable.

In particular, at least two authoritative resolvers that implement live signing had bugs when returning empty Resource Record sets for DNSSEC-signed zones, in combination with mixed-case queries. Mixed-case queries, also known as DNS 0x20, are used by some recursive resolvers to increase resilience against DNS poisoning attacks. DNSSEC-signing authoritative resolvers are expected to copy the same capitalization from the query into their ANSWER section, but sign the
response as if they had used all lowercase. In particular, PowerDNS
versions prior to 4.0.4 had this bug.

7. Differences versus RFC6844

This document obsoletes RFC6844. The most important change is to the
Certification Authority Processing section. RFC6844 specified an
algorithm that performed DNS tree-climbing not only on the FQDN being
processed, but also on all CNAMEs and DNAMEs encountered along the
way. This made the processing algorithm very inefficient when used
on FQDNs that utilize many CNAMEs, and would have made it difficult
for hosting providers to set CAA policies on their own FQDNs without
setting potentially unwanted CAA policies on their customers’ FQDNs.
This document specifies a simplified processing algorithm that only
performs tree climbing on the FQDN being processed, and leaves
processing of CNAMEs and DNAMEs up to the CA’s recursive resolver.

This document also includes a "Deployment Considerations" section
detailing experience gained with practical deployment of CAA
enforcement among CAs in the WebPKI.

This document clarifies the ABNF grammar for the issue and issuewild
tags and resolves some inconsistencies with the document text. In
particular, it specifies that parameters are separated with
semicolons. It also allows hyphens in Property Tags.

This document also clarifies processing of a CAA RRset that is not
empty, but contains no issue or issuewild tags.

This document removes the section titled "The CAA RR Type," merging
it with "Mechanism" because the definitions were mainly duplicates.
It moves the "Use of DNS Security" section into Security
Considerations. It renames "Certification Authority Processing" to
"Relevant Resource Record Set," and emphasizes the use of that term
to more clearly define which domains are affected by a given RRset.

8. IANA Considerations

IANA is requested to add [[[ RFC Editor: Please replace with this RFC ]]] as a reference for the Certification Authority Restriction Flags
and Certification Authority Restriction Properties registries, and
update references to [RFC6844] within those registries to refer to
[[[ RFC Editor: Please replace with this RFC ]]]. IANA is also
requested to update the CAA TYPE in the DNS Parameters registry with
a reference to [[[ RFC Editor: Please replace with this RFC ]]].
9. Acknowledgements

The authors would like to thank the following people who contributed to the design and documentation of this work item: Corey Bonnell, Chris Evans, Stephen Farrell, Jeff Hodges, Paul Hoffman, Tim Hollebeek, Stephen Kent, Adam Langley, Ben Laurie, James Manger, Chris Palmer, Scott Schmit, Sean Turner, and Ben Wilson.

10. References

10.1. Normative References


10.2. Informative References


Authors' Addresses

Phillip Hallam-Baker

Email: phill@hallambaker.com
Abstract

With the widespread adoption of post-quantum cryptography will come the need for an entity to possess multiple public keys on different cryptographic algorithms. Since the trustworthiness of individual post-quantum algorithms is at question, a multi-key cryptographic operation will need to be performed in such a way that breaking it requires breaking each of the component algorithms individually. This requires defining new structures for holding composite public keys and composite signature data.

This document defines the structures CompositePublicKey, CompositeSignatureValue, and CompositeParams, which are sequences of the respective structure for each component algorithm. This document also defines algorithms for generating and verifying composite signatures. This document makes no assumptions about what the component algorithms are, provided that their algorithm identifiers and signature generation and verification algorithms are defined.

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 January 5, 2020.
1. Introduction

During the transition to post-quantum cryptography, there will be uncertainty as to the strength of cryptographic algorithms; we will no longer fully trust traditional cryptography such as RSA, Diffie-Hellman, DSA and their elliptic curve variants, but we will also not fully trust their post-quantum replacements until they have had sufficient scrutiny. Unlike previous cryptographic algorithm migrations, the choice of when to migrate and which algorithms to migrate to, is not so clear. Even after the migration period, it may be advantageous for an entity’s cryptographic identity to be composed of multiple public-key algorithms.

The deployment of composite public keys and composite signatures using post-quantum algorithms will face two challenges:

- **Algorithm strength uncertainty**: During the transition period, some post-quantum signature and encryption algorithms will not be fully trusted, while also the trust in legacy public key algorithms will also start to erode. A relying party may learn some time after deployment that a public key algorithm has become untrustworthy, but in the interim, they may not know which algorithm an adversary has compromised.

- **Backwards compatibility**: During the transition period, post-quantum algorithms will not be supported by all clients.

This document provides a mechanism to address algorithm strength uncertainty by providing formats for encoding multiple public keys and multiple signature values into existing public key and signature fields, as well as an algorithm for validating a composite signature. The issue of backwards compatibility is left open to be addressed in separate draft(s).

This document is intended for general applicability anywhere that public key structures or digital signatures are used within PKIX structures.

EDNOTE: While the scope of this document is restricted to signatures, we note that the same "CompositePublicKey" structure is equally applicable to asymmetric encryption keys. Though a word of warning that the corresponding "encrypt / decrypt with a composite public key" logic is somewhat less obvious; a naive implementer might be tempted to follow the same pattern as below and encrypt the message with each public key separately and then concatenate the ciphertexts, which is wrong, they need to be nested. Specifying the correct implementation of such an encryption scheme is out of scope for this
document, but would be good work for someone in the standards community to pick up.

1.1. Terminology

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.

The following terms are used in this document:

ALGORITHM:
An information object class for identifying the type of cryptographic operation to be performed. This document is primarily concerned with algorithms for producing digital signatures, though the public key structure could just as easily hold encryption keys.

BER:
Basic Encoding Rules (BER) as defined in [X.690].

COMPONENT ALGORITHM:
A single basic algorithm which is contained within a composite algorithm.

COMPOSITE ALGORITHM:
An algorithm which is a sequence of one or more basic algorithm, as defined in Section 2.

DER:
Distinguished Encoding Rules as defined in [X.690].

PUBLIC / PRIVATE KEY:
The public and private portion of an asymmetric cryptographic key, making no assumptions about which algorithm.

PRIMITIVE PUBLIC KEY / SIGNATURE:
A public key or signature object of a non-composite algorithm type.

SIGNATURE:
A digital cryptographic signature, making no assumptions about which algorithm.
2. Composite Structures

In order for public keys and signatures to be composed of multiple algorithms, we define encodings consisting of a sequence of public key and signature primitives (aka "component algorithms") such that these structures can be used as a drop-in compatible way with existing public key or signature fields such as those found in PKCS#10 [RFC2986], CMP [RFC4210], X.509 [RFC5280], CMS [RFC5652].

This section defines the following structures:

- The id-alg-composite is an OID identifying a composite public key or signature object.
- The CompositePublicKey carries all the public keys associated with an identity within a single public key structure.
- The CompositePrivateKey carries all the private keys associated with an identity within a single private key structure.
- The CompositeSignatureValue, carries a sequence of signatures that are generated by a CompositePrivateKey, and can be verified with the corresponding compositePublicKey.

EDNOTE: the choice to define composite algorithm parameters as a sequence inside the existing fields avoids the exponential proliferation of OIDs that are needed for each pairwise combination of signature algorithms in other schemes for achieving multi-key certificates. This scheme also naturally extends from 2-keypair to n-keypair keys and certificates.

2.1. Algorithm Identifier

The same algorithm identifier is used for identifying a public key, a private key, and a signature. Additional encoding information is provided below for each of these objects.

```
id-alg-composite OBJECT IDENTIFIER ::= {
    iso(1)  identified-organization(3) dod(6) internet(1) private(4)
    enterprise(1) OpenCA(18227) Algorithms(2) id-alg-composite(1) }
```

EDNOTE: this is a temporary OID for the purposes of prototyping. We are requesting IANA to assign a permanent OID, see Section 5.
2.2. Composite Keys

A composite key is a single key object that performs an atomic signature or verification operation, using its encapsulated sequence of component keys.

The ASN.1 algorithm object for composite public and private keys is:

pk-Composite PUBLIC-KEY ::= {
    IDENTIFIER id-alg-composite
    KEY CompositePublicKey
    PARAMS ARE absent
    CERT-KEY-USAGE
        { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    PRIVATE-KEY CompositePrivateKey
}

EDNOTE1: the authors are currently unsure whether the params should be absent (ie this structure simply says "I am a composite algorithm"), or used to duplicate some amount of information about what the component algorithms are. See Section 2.3 for a longer ENDOTE on this.

EDNOTE2: In order to reduce complexity, we are intentionally limiting the scope of this draft to signature-type CERT-KEY-USAGES, but we note that it would be trivial to extend it to encryption-type keys.

2.2.1. Key Usage Bits

The intended application for the key is indicated in the keyUsage certificate extension and defined in the CERT-KEY-USAGE field of pk-Composite.

If the keyUsage extension is present in an end-entity certificate that indicates id-alg-composite, then the keyUsage extension MUST contain one or both of the following values:

    nonRepudiation; and
digitalSignature.

If the keyUsage extension is present in a certification authority certificate that indicates id-alg-composite, then the keyUsage extension MUST contain one or more of the following values:

    nonRepudiation;
digitalSignature;
keyCertSign; and
cRLSign.
As this draft only covers composite signatures, the key usage bits specified here apply to all component keys within a composite key.

2.3. Composite Public Key

Composite public key data is represented by the following structure:

CompositePublicKey ::= SEQUENCE SIZE (1..MAX) OF SubjectPublicKeyInfo

The corresponding AlgorithmIdentifier for a composite public key MUST use the id-alg-composite object identifier, defined in Section 2.1, and the parameters field MUST be absent.

A composite public key MUST contain at least one component public key.

A CompositePublicKey MUST NOT contain a component public key which itself describes a composite key; i.e. recursive CompositePublicKeys are not allowed.

Each element of a CompositePublicKey is a SubjectPublicKeyInfo object one of the component public keys. When the CompositePublicKey must be provided in octet string or bit string format, the data structure is encoded as specified in Section 2.6.

--- Begin EDNOTE ---

EDNOTE: there has been a fair amount of discussion among the authors about whether the component public key should contain a full SubjectPublicKeyInfo for each component algorithm, or whether the (algID, and algParams) should be move to the params of the PUBLIC-KEY or OID, and only the BIT STRINGs of the component public key values contained in the CompositePublicKey.

Using a wonky, simplified notation, the alternatives considered were:
Current composite:
CompositeAlg: {
    algorithm=(id-alg-composite, none)
    subjectPublicKey=SEQ SPKI[{{algID1, algParams1}, value1},
    SPKI[{{algID2, algParams2}, value2}, ..]}
}

Alternative 1:
CompositeAlg: {
    algorithm=(id-alg-composite, {{algID1, algParams1},
    {algID2, algParams2}, ..})
    subjectPublicKey=SEQ BIT STRING[value1, value2, ..]}

Alternative 2:
CompositeAlg: {
    algorithm=(id-alg-composite, {algID1, algID2, ..})
    subjectPublicKey=SEQ SPKI[{{algID1, algParams1}, value1},
    {algID2, algParams2}, value2], ..]}

The authors have decided, for the time being, to use the current approach since it A) promotes ease of modifying existing software whose APIs require SubjectPublicKeyInfos to be passed, and B) avoids bloating wire protocols with duplicated information.

We note that the chosen approach means that the algorithm field essentially carries no useful information about the key it’s describing. Analysis is required to see if there are any circumstances in which this opens up cryptographic attacks, such as algorithm substitution or stripping attacks. --- End EDNOTE ---

2.4. Composite Private Key

The composite private key data is represented by the following structure:

CompositePrivateKey ::= SEQUENCE SIZE (1..MAX) OF OneAsymmetricKey

Each element is a OneAsymmetricKey [RFC5958] object for a component private key.

The corresponding AlgorithmIdentifier for a composite private key MUST use the id-alg-composite object identifier, and the parameters field MUST be absent.
A CompositePrivateKey MUST contain at least one component private key, and they MUST be in the same order as in the corresponding CompositePublicKey.

2.5. Composite Signature

The ASN.1 algorithm object for a composite signature is:

sa-CompositeSignature SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-alg-composite
  VALUE CompositeSignatureValue
  PARAMS TYPE CompositeParams ARE required
  PUBLIC-KEYS { pk-Composite }
  SMIME-CAPS { IDENTIFIED BY id-alg-composite } }

The id-alg-composite object identifier MUST be used to identify when a signature has been created by a composite private key, and the following algorithm parameters MUST be included:

CompositeParams ::= SEQUENCE SIZE (1..MAX) OF AlgorithmIdentifier

The signature’s CompositeParams sequence MUST contain the same component algorithms listed in the same order as in the associated CompositePrivateKey and CompositePublicKey.

The output of the composite signature algorithm is the DER encoding of the following structure:

CompositeSignatureValue ::= SEQUENCE SIZE (1..MAX) OF BIT STRING

Where each BIT STRING within the SEQUENCE is a signature value produced by one of the component keys. It MUST contain MUST contain one signature value produced by each component key, and in the same order as in the associated "CompositeParams", CompositePublicKey, and CompositePrivateKey objects.

The choice of "SEQUENCE OF BIT STRING", rather than for example a single BIT STRING containing the concatenated signature values, is to gracefully handle variable-length signature values by taking advantage of ASN.1’s build-in length fields.

2.6. Encoding Rules

Many protocol specifications will require that the composite public key, composite private key, and composite signature data structures be represented by an octet string or bit string.
When an octet string is required, the DER encoding of the composite data structure SHALL be used directly.

When a bit string is required, the octets of the DER encoded composite data structure SHALL be used as the bits of the bit string, with the most significant bit of the first octet becoming the first bit, and so on, ending with the least significant bit of the last octet becoming the last bit of the bit string.

In the interests of simplicity and avoiding compatibility issues, implementations that parse these structures MAY accept both BER and DER.

3. Composite Signature Algorithm

This section specifies the algorithms for generating and verifying composite signatures.

This algorithm addresses algorithm strength uncertainty by providing the verifier with parallel signatures from all the component signature algorithms; thus breaking the composite signature would require breaking all of the component signatures.

3.1. Composite Signature Generation

Generation of a composite signature involves applying each component algorithm’s signature routine to the input message according to its specification, and then placing each component signature value into the "CompositeSignatureValue" structure defined in Section 2.5.

The following algorithm is used to generate composite signature values.
Input:
    K1, K2, .., Kn Private keys for the n component signature algorithms
    M Message to be signed, an octet string

Output:
    S The signature, a CompositeSignatureValue

Signature Generation Procedure:
1. Generate the n component signatures independently, according to their algorithm specifications.
   
   for i := 1 to n
       Si := Sign( Ki, M )

2. Encode each component signature S1, S2, .., Sn into a BIT STRING according to its algorithm specification.
   
   S ::= Sequence { S1, S2, .., Sn }

3. Output S

Since recursive composite public keys are disallowed in Section 2.3, no component signature may itself be composite; ie the signature generation routine MUST fail if one of the private keys K1, K2, .., Kn is composite with the OID id-alg-composite.

A composite signature MUST produce and include in the output a signature value for every component key in the corresponding CompositePublicKey.

EDNOTE1: With NIST’s position that they will standardize use-case-specific algorithm suites, the authors are aware of potential use-cases where a PKI entity may want to have many public keys, but only sign with a subset for each signature. At the present time, this draft does not allow for this because the algorithm for verifying "subset-signatures" in a way that is secure against algorithm stripping attacks would be very complex and prone to implementation errors (currently, the verifier can detect omitted signatures even if it does not recognize all the algorithm OIDs because the count will be wrong. In a subset-signature algorithm, additional mechanisms would be needed to specify for each component key, whether it is meant to produce a signature or not). The draft-compliant way to achieve a "subset-signature" behaviour would be for each PKI entity to have multiple public keys (and certificates) with overlapping subsets of their component keys. We welcome public opinions on whether this is sufficient, or whether this draft should specify a subset-signature algorithm.

[Page 11]
EDNOTE2: The authors are also aware of a potential use-case of combining signature and KEM keys inside a single public key / certificate. This would give us back the "dual-usage key" property that was so appealing about RSA. At the present time, this draft does not allow for this because, again, the algorithm for verifying "subset-signatures" in a secure way would be very complex. We also welcome public opinions on this.

3.2. Composite Signature Verification

Verification of a composite signature involves applying each component algorithm’s verification routine according to its specification, and then outputting "Valid signature" (true) if a sufficient number of component algorithms were valid, and "Invalid signature" (false) otherwise.

In order to future-proof implementations of verifiers against evolutions in cryptographic algorithms and attacks against them, implementations SHOULD include a field-updatable policy mechanism for determining which and/or how many component algorithms must be valid in order for the composite signature as a whole to be considered valid. This section assumes the existence of such a policy mechanism, denoted as "checkPolicy(A1, A2, ..., An)" in the algorithm below. The implementation of such a policy mechanism is largely the responsibility of the verifier / client and therefore is out of scope for this document, but at a minimum, one component signature MUST be recognized and validated for the composite signature to be considered valid.

Modifications of the provided verification algorithm are permitted, so long as they are strengthening, and not weakening, this algorithm. In other words, any modified versions of this algorithm MUST return "Invalid signature" whenever the sample algorithm does, with the one exception noted below.
Input:
P    Signer’s composite public key
M    Message whose signature is to be verified, an octet string
S    Composite Signature to be verified
A    Composite Algorithm identifier

Output:
Validity    "Valid signature" (true) if the composite signature is valid, "Invalid signature" (false) otherwise.

Signature Verification Procedure:
1. Parse P, S, A into the component public keys, signatures, and algorithm identifiers

   P1, P2, .., Pn := Desequence( P )
   S1, S2, .., Sn := Desequence( S )
   A1, A2, .., An := Desequence( A )

   If Error during Desequencing, or the three sequences have different numbers of elements, then output "Invalid signature" and stop.

2. Check client policy to see whether A1, A2, .., An constitutes an acceptable combination of algorithms.

   if not checkPolicy(A1, A2, .., An), then output "Invalid signature"

3. Check each component signature individually, according to its algorithm specification.
   If any fail, then the entire signature validation fails.

   for i := 1 to n
      if not verify( Pi, M, Si ), then output "Invalid signature"

   if all succeeded, then output "Valid signature"

Since recursive composite public keys are disallowed in Section 2.3, no component signature may be composite; ie the signature verification procedure MUST fail if any of the public keys P1, P2, .., Pn or algorithm identifiers A1, A2, .., An are composite with the OID id-alg-composite.

Exception to this algorithm: There will be circumstances in which the verifier does not have cryptographic libraries for all of the provided component algorithms, or where the performance gains from
omitting algorithms justifies the loss of security. In these cases, an acceptable modification to this algorithm is to produce in step 2 one or more subsets of the algorithms "A1, A2, ..., An" which constitute acceptable combinations, outputting "Invalid signature" if an acceptable subset can not be found, and then in step 3 only perform verification of the necessary component algorithms.

Implementations SHOULD verify all recognized and supported algorithms, and output "Invalid signature" if the verification of any component signature fails, but MAY choose to only verify a subset of the algorithms for the reasons stated above.

4. In Practice

This section addresses practical issues of how this draft affects other protocols and standards.

```--- BEGIN EDNOTE ---
EDNOTE: Possible topics to address:

- The size of these certs and cert chains.
- In particular, implications for (large) composite keys / signatures / certs on the handshake stages of TLS and IKEv2.
- If a cert in the chain is a composite cert then does the whole chain need to be of composite Certs?
- We could also explain that the root CA cert does not have to be of the same algorithms. The root cert SHOULD NOT be transferred in the authentication exchange to save transport overhead and thus it can be different than the intermediate and leaf certs.
- We could talk about overhead (size and processing).
- We could also discuss backwards compatibility.
- We could include a subsection about implementation considerations.

--- END EDNOTE ---
```

4.1. PEM Storage of Composite Private Keys

CompositePrivateKeys can be encoded to the PEM format by placing a CompositePrivateKey into the privateKey field of a PrivateKeyInfo or OneAsymmetricKey object, and then applying the PEM encoding rules as
defined in [RFC7468] section 10 and 11 for plaintext and encrypted private keys, respectively.

EDNOTE: Do we really need this? Isn’t it obvious?

4.2. Asymmetric Key Packages (CMS)

The Cryptographic Message Syntax (CMS), as defined in [RFC5652], can be used to digitally sign, digest, authenticate, or encrypt the asymmetric key format content type.

When encoding composite private keys, the privateKeyAlgorithm in the OneAsymmetricKey SHALL be set to id-alg-composite.

The parameters of the privateKeyAlgorithm SHALL be a sequence of AlgorithmIdentifier objects, each of which are encoded according to the rules defined for each of the different keys in the composite private key.

The value of the privateKey field in the OneAsymmetricKey SHALL be set to the DER encoding of the SEQUENCE of private key values that make up the composite key. The number and order of elements in the sequence SHALL be the same as identified in the sequence of parameters in the privateKeyAlgorithm.

The value of the publicKey (if present) SHALL be set to the DER encoding of the corresponding CompositePublicKey. If this field is present, the number and order of component keys MUST be the same as identified in the sequence of parameters in the privateKeyAlgorithm.

The value of the attributes is encoded as usual.

4.3. Cryptographic protocols

This section talks about how protocols like (D)TLS and IKEv2 are affected by this specifications. It will not attempt to solve all these problems, but it will explain the rationale, how things will work and what open problems need to be solved. Obvious issues that need to be discussed.

- How does the protocol declare support for composite signatures? TLS has hooks for declaring support for specific signature algorithms, however it would need to be extended, because the client would need to declare support for both the composite infrastructure, as well as for the various component signature algorithms.
o How does the protocol use the multiple keys. The obvious way would be to have the server sign using its composite public key; is this sufficient.

o Overhead; including certificate size, signature processing time, and size of the signature.

o How to deal with crypto protocols that use public key encryption algorithms; this document only lists how to work with signature algorithms. Encoding composite public keys is straightforward; encoding composite ciphertexts is less so - we decided to put that off to another draft.

5. IANA Considerations

The ASN.1 module OID is TBD. The id-alg-composite OID is to be assigned by IANA. The authors suggest to use the id-pkix arc for this usage:

id-alg-composite OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) algorithms(6) composite(??) }

6. Security Considerations

6.1. Policy for Deprecated and Acceptable Algorithms

Traditionally, a public key, certificate, or signature contains a single cryptographic algorithm. If and when an algorithm becomes deprecated (for example, RSA-512, or SHA1), it is obvious that structures using that algorithm are implicitly revoked.

In the composite model this is less obvious since a single public key, certificate, or signature may contain a mixture of deprecated and non-deprecated algorithms. Moreover, implementers may decide that certain cryptographic algorithms have complementary security properties and are acceptable in combination even though neither algorithm is acceptable by itself.

In Section 3.2, we specify that the signature verification routine must include a step to check that the combination of algorithms is acceptable under local policy:

2. Check policy to see whether A1, A2, ..., An constitutes a valid combination of algorithms.
   if not checkPolicy(A1, A2, ..., An), then output "Invalid signature"
While intentionally not specified in this document, implementors should put careful thought into implementing a meaningful policy mechanism within the context of their signature verification engines.

6.2. Protection of Private Keys

This structures described in this document do not protect the private keys information in any way unless combined with a security protocol or encryption properties of the objects (if any) where the CompositePrivateKey is used (see next Section).

Protection of the private key information is vital to public key cryptography. The consequences of disclosure depend on the purpose of the private key. If a private key is used for signature, then the disclosure allows unauthorized signing. If a private key is used for key management, then disclosure allows unauthorized parties to access the managed keying material. The encryption algorithm used in the encryption process must be as ‘strong’ as the key it is protecting.

6.3. Checking for Compromised Key Reuse

CA implementations need to be careful when checking for compromised key reuse, for example as required by WebTrust regulations; when checking for compromised keys, you MUST unpack the CompositePublicKey structure and compare individual component keys.

6.4. Composite Encryption and KEMs

This document deals only with signature keys. While the CompositePublicKey and CompositePrivateKey structures could equally be used to hold encryption or KEM keys, the authors warn that there are non-trivial design decisions to be made when constructing a multi-key public key encryption or KEM algorithm. Some of these design and implementation decisions, if done incorrectly will result in a catastrophic loss of security. We leave it to the community to standardize analogous composite encryption and KEM schemes.

7. Appendices

7.1. ASN.1 Module

<CODE STARTS>

Composite-Signatures-2019
{ TBD }

DEFINITIONS IMPLICIT TAGS ::= BEGIN
EXPORTS ALL;

IMPORTS
PUBLIC-KEY, SIGNATURE-ALGORITHM
FROM AlgorithmInformation-2009  -- RFC 5912 [X509ASN1]
{ iso(1) identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) id-mod(0)
  id-mod-algorithmInformation-02(58) }

SubjectPublicKeyInfo
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) }

OneAsymmetricKey
FROM AsymmetricKeyPackageModuleV1
{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
  pkcs-9(9) smime(16) modules(0)
  id-mod-asymmetricKeyPkgV1(50) };

--
-- Object Identifiers
--

id-alg-composite OBJECT IDENTIFIER ::= { TBD }

--
-- Public Key
--

pk-Composite PUBLIC-KEY ::= {
  IDENTIFIER id-alg-composite
  KEY CompositePublicKey
  PARAMS ARE absent
  CERT-KEY-USAGE
  { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
  PRIVATE-KEY CompositePrivateKey
}

CompositePublicKey ::= SEQUENCE SIZE (1..MAX) OF SubjectPublicKeyInfo

CompositePrivateKey ::= SEQUENCE SIZE (1..MAX) OF OneAsymmetricKey

--
-- Signature Algorithm
--
sa-CompositeSignature SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-alg-composite
  VALUE CompositeSignatureValue
  PARAMS TYPE CompositeParams ARE required
  PUBLIC-KEYS { pk-Composite }
  SMIME-CAPS { IDENTIFIED BY id-alg-composite } }

CompositeParams ::= SEQUENCE SIZE (1..MAX) OF AlgorithmIdentifier

CompositeSignatureValue ::= SEQUENCE SIZE (1..MAX) OF BIT STRING

END

<CODE ENDS>

7.2. Intellectual Property Considerations

The authors are aware that Massimiliano Pala and CableLabs have applied for Intellectual Property Rights around composite key, signatures, and certificates. We have a verbal agreement with Max that this IP will be made freely available to the community.

As of this version of the draft, the authors have reviewed and provided feedback on the March 24, 2019 version of the IPR disclosure, available at https://datatracker.ietf.org/ipr/3481/, and are awaiting the posting of an updated version that covers this draft.

EDNOTE: remove this section once the IPR disclosure is posted and tagged against this draft.

8. Contributors and Acknowledgements

This document incorporates contributions and comments from a large group of experts. The Editors would especially like to acknowledge the expertise and tireless dedication of the following people, who attended many long meetings and generated millions of bytes of electronic mail and VOIP traffic over the past year in pursuit of this document:

John Gray (Entrust Datacard), Serge Mister (Entrust Datacard), Scott Fluhrer (Cisco Systems), Panos Kampanakis (Cisco Systems), Daniel Van Geest (ISARA), and Tim Hollebeek (Digicert).

We are grateful to all, including any contributors who may have been inadvertently omitted from this list.
This document borrows text from similar documents, including those referenced below. Thanks go to the authors of those documents. "Copying always makes things easier and less error prone" - [RFC8411].

9. References

9.1. Normative References


9.2. Informative References

[I-D.pala-composite-crypto]

[I-D.truskovsky-lamps-pq-hybrid-x509]


Authors’ Addresses

Mike Ounsworth
Entrust Datacard Limited
1000 Innovation Drive
Ottawa, Ontario K2K 1E3
Canada

Email: mike.ounsworth@entrustdatacard.com

Massimiliano Pala
CableLabs

Email: director@openca.org
Abstract

This document updates RFC 5480 to specify semantics for the keyEncipherment and dataEncipherment key usage bits when used in certificates that support Elliptic Curve Cryptography.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 14, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

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

[RFC5480] specifies the syntax and semantics for the Subject Public Key Information field in certificates that support Elliptic Curve Cryptography. As part of these semantics, it defines what combinations are permissible for the values of the key usage extensions [RFC5280]. [RFC5480] specifies 7 of the 9 values; it makes no mention of keyEncipherment and dataEncipherment key usage bits. This document corrects this omission, by updating Section 3 of [RFC5480] to make it clear that neither keyEncipherment nor the dataEncipherment key usage bits are set for key agreement algorithms.

2. Terminology

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.

3. Updates to Section 3

If the keyUsage extension is present in a certificate that indicates id-ecPublicKey as algorithm of AlgorithmIdentifier [RFC2986] in SubjectPublicKeyInfo, then following values MUST NOT be present:

- keyEncipherment;
- dataEncipherment.

If the keyUsage extension is present in a certificate that indicates id-ecDH or id-ecMQV in SubjectPublicKeyInfo, then the following values also MUST NOT be present:

- keyEncipherment;
- dataEncipherment.
4. Security Considerations

This document introduces no new security considerations beyond those found in [RFC5480].

5. IANA Considerations

This document makes no request of IANA.

6. Normative References


Authors’ Addresses

Tadahiko Ito
SECOM CO., LTD.

Email: tadahiko.ito.public@gmail.com

Sean Turner
sn3rd

Email: sean@sn3rd.com