



BSI Standards Publication

Information technology — Generic applications of ASN.1

Part 4: Cryptographic message syntax

National foreword

This British Standard is the UK implementation of ISO/IEC 24824-4:2021.

The UK participation in its preparation was entrusted to Technical Committee ICT/1, Information systems co-ordination.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2021
Published by BSI Standards Limited 2021

ISBN 978 0 539 12107 0

ICS 35.100.60

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 April 2021.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

**Information technology — Generic
applications of ASN.1 —**

**Part 4:
Cryptographic message syntax**





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2021

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier; Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, www.iec.ch/understanding-standards.

This document was prepared by ITU-T [Telecommunication Standardization Sector of ITU] (as ITU-T X.894 [10/2018]) and drafted in accordance with its editorial rules, in collaboration with Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 6, *Telecommunications and information exchange between systems*.

A list of all parts in the ISO/IEC 24824 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

CONTENTS

	<i>Page</i>
1 Scope	1
2 Normative references	1
2.1 Identical Recommendations International Standards	1
2.2 Paired Recommendations International Standards equivalent in technical content	1
2.3 Additional References	1
3 Definitions	1
4 Abbreviations	2
5 Conventions	2
6 Cryptographic message syntax	2
7 Signcryption	3
7.1 The SigncrypteData type	4
7.2 The ContentInformation type	4
7.3 The Signcrypter type	10
8 Quantum safe SignedData signatures	11
8.1 Detached content consideration	12
8.2 Time stamp consideration	12
8.3 The tokenizedParts attribute	13
9 Other key management techniques	13
9.1 Constructive key management	13
9.2 Database encryption key management	14
Annex A – ASN.1 modules	17
A.1 Main CMS module (from IETF RFC 6268)	17
A.2 Module CMSObjectIdentifiers	23
A.3 Module AlgorithmInformation-2009 (from IETF RFC 5912)	25
A.4 Module CryptographicMessageSyntaxAlgorithms-2009 (from IETF RFC 5911)	32
A.5 Module PKIX-Algs-2009 (from IETF RFC 5912)	35
A.6 Module PKIXAttributeCertificate-2009 (from IETF RFC 5912)	42
A.7 Module AttributeCertificateVersion1-2009 (from IETF RFC 5912)	46
A.8 Module PKIX-CommonTypes-2009 (from IETF RFC 5912)	47
A.9 Module PKIX-X400Address-2009 (from IETF RFC 5912)	50
A.10 Module PKIX1Explicit-2009 (from IETF RFC 5912)	54
A.11 Module PKIXImplicit-2009 (from IETF RFC 5912)	60
A.12 Module PKIX1-PSS-OAEP-Algorithms-2009 (from IETF RFC 5912)	67
A.13 Module SecureMimeMessageV3dot1-2009 (from IETF RFC 5911)	71
A.14 Module CMSSigncryption	73
A.15 Module CMSCKMKeyManagement	75
A.16 Module CMSDBKeyManagement	77
A.17 Module CMSProfileAttributes	79
A.18 Module TokenizationManifest	80
A.19 Module TransientKey	81
A.20 Module TrustedTimestamp	83
A.21 Module ANSI-X9-42	88
A.22 Module ANSI-X9-62	91
Annex B – Object identifiers defined in this Recommendation International Standard	96
Bibliography	97

INTERNATIONAL STANDARD ISO/IEC 24824-4 RECOMMENDATION ITU-T X.894

Information technology — Generic applications of ASN.1 —

Part 4:

Cryptographic message syntax

1 Scope

This Recommendation | International Standard enhances the existing cryptographic message syntax (CMS) protocol by adding signcryption techniques and providing a new Abstract Syntax Notation One (ASN.1) module which conforms to the latest edition of the ASN.1 standard which can be used with all standardized encoding rules of ASN.1.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- Recommendation ITU-T X.509 (2016) | ISO/IEC 9594-8:2017, *Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks*.

2.2 Paired Recommendations | International Standards equivalent in technical content

None.

2.3 Additional References

- ISO 11568-1:2005, *Banking – Key management (retail) – Part 1: Principles*.
- ISO/IEC 11770-6:2016, *Information technology – Security techniques – Key management – Part 6: Key derivation*.
- ISO/IEC 18033-2:2006, *Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers*.
- ISO/IEC 29150:2011, *Information technology – Security techniques – Signcryption*.
- IETF RFC 5652 (2009), *Cryptographic message syntax (CMS)*.
- IETF RFC 6268 (2011), *Additional new ASN.1 modules for the cryptographic message syntax (CMS) and the public key infrastructure using X.509 (PKIX)*.

3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply:

The following terms are defined in Rec. ITU-T X.509 | ISO/IEC 9594-8:

- a) attribute certificate;
- b) CA certificate;
- c) certificate revocation list.

The following term is defined in ISO/IEC 29150:

- signcryption

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply:

ASN.1	Abstract Syntax Notation One
CEK	Content Encryption Key
CKM	Constructive Key Management
CMS	Cryptographic Message Syntax
CRL	Certificate Revocation List
DBEKM	Database Encryption Key Management
DK	Data encryption Key
HK	HMAC Key
HMAC	Hashed Message Authentication
ID	Identifier
KDF	Key Derivation Function
MK	Master Key encryption key
PBKDF	Password-Based KDF
SCD	Secure Cryptographic Device
SHA	Secure Hash Algorithm
URI	Uniform Resource Identifier
XML	extensible Markup Language

5 Conventions

None.

6 Cryptographic message syntax

CMS is defined in the base text, IETF RFC 5652. ASN.1 modules have been revised to conform to the current ASN.1 standard in IETF RFC 6268.

CMS defines the following content types:

- data: used to transfer data defined string of octets;
- signed data: used to transfer data with zero or more signatures;
- enveloped data: used to transfer encrypted data with one or more content-encryption keys;
- digested data: used to transfer data with a message digest;
- encrypted data: used to transfer encrypted data;
- authenticated data: used to transfer data with a message authentication code and one or more encrypted authentication keys.

Each of these content types is uniquely identified by an object identifier:

- for data:
**id-data OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840)
rsadsi(113549)pkcs(1) pkcs7(7) 1}**
- for signed data:
**id-signedData OBJECT IDENTIFIER ::= {iso(1) member-body(2)
us(840)rsadsi(113549) pkcs(1) pkcs7(7) 2}**
- for enveloped data:

- ```

id-envelopedData OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840)
 rsadsi(113549) pkcs(1) pkcs7(7) 3}
- for digested data:
id-digestedData OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840)
 rsadsi(113549) pkcs(1) pkcs7(7) 5}
- for encrypted data:
id-encryptedData OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840)
 rsadsi(113549) pkcs(1) pkcs7(7) 6}
- for authenticated data:
id-ct-authData OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840)
 rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) ct(1) 2}

```

Data transferred with CMS use the following ASN.1 type:

```

ContentInfo ::= SEQUENCE {
 contentType CONTENT-TYPE.&id({ContentSet}),
 content [0] EXPLICIT CONTENT-
 TYPE.&Type({ContentSet}{@contentType})}

```

The **CONTENT-TYPE** information object class is defined as **TYPE-IDENTIFIER** and is used to assign one of the previous object identifiers to the corresponding ASN.1 type.

**CONTENT-TYPE ::= TYPE-IDENTIFIER**

**ContentType ::= CONTENT-TYPE.&id**

```

ContentSet CONTENT-TYPE ::= {
 -- Define the set of content types to be recognized
 ct-Data |
 ct-SignedData |
 ct-EnvelopedData |
 ct-DigestedData |
 ct-EncryptedData |
 ct-AuthenticatedData,
 ...}

```

|                             |                                                                          |
|-----------------------------|--------------------------------------------------------------------------|
| <b>ct-Data</b>              | <b>CONTENT-TYPE ::= {OCTET STRING IDENTIFIED BY id-data}</b>             |
| <b>ct-SignedData</b>        | <b>CONTENT-TYPE ::= {SignedData IDENTIFIED BY id-signedData}</b>         |
| <b>ct-EnvelopedData</b>     | <b>CONTENT-TYPE ::= {EnvelopedData IDENTIFIED BY id-envelopedData}</b>   |
| <b>ct-DigestedData</b>      | <b>CONTENT-TYPE ::= {DigestedData IDENTIFIED BY id-digestedData}</b>     |
| <b>ct-EncryptedData</b>     | <b>CONTENT-TYPE ::= {EncryptedData IDENTIFIED BY id-encryptedData}</b>   |
| <b>ct-AuthenticatedData</b> | <b>CONTENT-TYPE ::= {AuthenticatedData IDENTIFIED BY id-ct-authData}</b> |

Other content types can be defined by creation of new information objects of **CONTENT-TYPE** information object class using unique object identifiers.

The **ct-SignedCryptedData** defined in clause 7 is an example.

## 7 Signcryption

The **SigncryptData** uses the signcryption technique defined in ISO/IEC 29150. The signcryption technique simultaneously signs and encrypts the data to achieve origin authentication, data integrity and confidentiality. Signcryption can be used in CMS in four different modes:

- signcrypted-content**: content of any type or format is signcrypted using the signcryption algorithm;
- signcrypted-attributes**: content of any type or format and a collection of attributes of any type or format are together signcrypted;
- signcrypted-components**: elements of content of any type or format are signcrypted for one or more message recipients using the public-private keys of the sender and the public key of each recipient.

- d) **signcrypted-envelope**: a fresh symmetric key is used to encrypt content of any type or format and the resulting ciphertext is transmitted as an octet string.

## 7.1 The SigncryptData type

**id-signcryptData** OBJECT IDENTIFIER ::= {itu-t recommendation(0) x(24) cms-profile(894) signcrypt(1) data(0)}

**ct-SigncryptData** CONTENT-TYPE ::= {  
TYPE SigncryptData IDENTIFIED BY id-signcryptData}

**SigncryptData** ::= SEQUENCE {  
version CMSVersion,  
contentInformation ContentInformation,  
certificates CertificateSet OPTIONAL,  
crls RevocationInfoChoices OPTIONAL,  
signcrypters Signcrypters  
}

**Signcrypters** ::= SEQUENCE SIZE(1..MAX) OF Signcrypter

- version** is version number.
- contentInformation** identifies the signcrypted data processing mode.
- certificates** is a collection of public key or attribute certificates to facilitate the validation of the public keys of the signcrypters. This collection may contain more certificates than necessary or fewer and, in the latter case, recipients have to use other means to get other certificates.
- crls** is a collection of public key or attribute certificate revocation lists (CRLs) to facilitate the validation of the certificates of the signcrypters. As for **certificates**, this collection may contain more CRLs than necessary or fewer and, in the latter case, recipients have to use other means to get other CRLs.
- signcrypters** is a collection of all signcrypter specific information.

## 7.2 The ContentInformation type

**ContentInformation** ::= SEQUENCE {  
mode Mode,  
content Content OPTIONAL  
}

The modes are defined using the following information object class:

**MODE** ::= CLASS {  
    &Type OPTIONAL,  
    &id OBJECT IDENTIFIER UNIQUE  
}  
WITH SYNTAX {[WITH SYNTAX &Type] ID &id}

Four modes are currently defined:

- signcryptAttributes**: content and attributes of any type or format are signcrypted.
- signcryptComponents**: specified components of content of any type or format are signcrypted, and the resulting cryptogram is signed along with a manifest of signcrypted locations and other attributes.
- signcryptContent**: content of any type or format is signcrypted.
- signcryptEnveloped**: content of any type or format is encrypted under a symmetric key to create ciphertext for sharing with one or more message recipients. That symmetric key and a message digest of the ciphertext are signcrypted for each message recipient using the public-private keys of the sender and the public key of each recipient.

**Mode** ::= MODE.&id({ProcessingModes})

**ProcessingModes** MODE ::= {  
    signcryptAttributes |  
    signcryptComponents |  
    signcryptContent |

```

 signcryptEnveloped,
 ...-- Expect additional processing modes --
}

```

**Content** ::= OCTET STRING (SIZE (1..MAX))

### 7.2.1 The signcryptContent mode

In the signcrypt-content mode, content of any type or format is signcrypt using the signcrypt algorithm identified in the **signcryptDataAlgorithm** component of the **signcrypters** component of type **SigncryptData**. The message sender applies this signcrypt algorithm to the content using the sender public and private keys and the recipient public key. These keys are identified in the message as values of type **SigncrypterIDs** and the **signcryptValue** component of type **Signcrypter** contains the results of signcrypt.

In this processing mode of **SigncryptData**, there are no signcrypt attributes and the optional **signatureInformation** component is not present. The message sender at their option may include values in the **unsigncryptAttributes** component of the **Signcrypter** component for any recipient. The signcrypt-content mode is indicated in a message by the following information object identifier:

**signcryptContent MODE** ::= { ID signcrypt-content }

A value of type **SigncryptData** using signcrypt-content mode is created as follows:

- a) the value of the **version** component of type **SigncryptData** is set to 1;
- b) in the **contentInformation** component of type **SigncryptData**, set the value of mode to the signcrypt-content object identifier. the optional **content** component should not be present;
- c) optionally, include values in the **certificates** component of type **SigncryptData**;
- d) optionally, include values in the **crls** component of type **SigncryptData**;
- e) for each message recipient, include a value of type **Signcrypter** in the **signcrypters** component of type **SigncryptData** as follows:
  - i) the value of the **version** component of type **Signcrypter** is set to 1,
  - ii) in the **sids** component of type **Signcrypter** set the components of a value of type **SigncrypterIDs** to identify the public-private key pairs of the message sender and recipient,
  - iii) set the value of the **signcryptDataAlgorithm** component of type **Signcrypter** to identify the signcrypt algorithm and any associated parameters,
  - iv) include the results of **signcrypting** content of any type or format encoded as a value of type OCTET STRING in the **signcryptValue** component of type **Signcrypter**,
  - v) the optional **signatureInformation** component of type **Signcrypter** should not be present;
  - vi) optionally, include values in the **unsigncryptAttributes** component of type **Signcrypter**.

To recover the plaintext from the **SigncryptData** message, a message recipient should perform the following steps:

- 1) search the list of per recipient values of type **Signcrypter** in the **signcrypters** component of type **SigncryptData** to locate the recipient public-private key pair in the **sids** component of type **Signcrypter**;
- 2) designcrypt the value in the **signcryptValue** component of type **Signcrypter** for a given recipient using their public-private key pair of the recipient and the public key of the sender identified in the **sids** component of type **Signcrypter**, and the signcrypt algorithm and associated parameters provided in the **signcryptDataAlgorithm** component of type **Signcrypter**.

Perform certificate path validation to gain assurance that the sender public key certificate is trusted.

### 7.2.2 The signcryptAttributes mode

In the signcrypt-attributes mode, content of any type or format and a collection of attributes of any type or format in the **signcryptAttributes** component of type **SigncrypterInfo** are together signcrypt as specified for the signcrypt-content mode. At least three attributes shall be present: the messageDigest, the **contentType**, and the **signcryptAttributes** attribute.

The signcrypted-attributes mode is indicated in a message by the following information object identifier:

**signcryptedAttributes MODE ::= { ID signcrypted-attributes }**

A value of type **SigncryptedExceptionData** using signcrypted-attributes mode is created as follows:

- a) the value of the **version** component of type **SigncryptedExceptionData** is set to 1;
- b) in the **contentInformation** component of type **SigncryptedExceptionData**, set the value of mode to the **signcrypted-attributes** object identifier. the optional **content** component should not be present;
- c) optionally, include values in the **certificates** component of type **SigncryptedExceptionData**;
- d) optionally, include values in the **crls** component of type **SigncryptedExceptionData**;
- e) for each message recipient, include a value of type **SigncryptedException** in the **signcryptedExceptions** component of type **SigncryptedExceptionData** as follows:
  - i) the value of the **version** component of type **SigncryptedException** is set to 1;
  - ii) in the **sids** component of type **SigncryptedException** set the components of a value of type **SigncryptedExceptionIDs** to identify the public-private key pairs of the message sender and recipient;
  - iii) set the value of the **signcryptedExceptionAlgorithm** component of type **SigncryptedException** to identify the signcryption algorithm and any associated parameters;
  - iv) create a value of type **ToBeSigncryptedException** by setting its **content** component to a value of type **Content** and its **attributes** component to a value of type **SigncryptedExceptionAttributes**;
  - v) signcrypt an encoded value of type **ToBeSigncryptedException** and include the results of the processing in the **signcryptionValue** component of type **SigncryptedException**;
  - vi) the optional **signatureInformation** component of type **SigncryptedExceptionInfo** should not be present;
  - vii) optionally, include values in the **unsigncryptedExceptionAttributes** component of type **SigncryptedException**.

To recover the plaintext from the **SigncryptedExceptionData** message, a message recipient should perform the following steps:

- a) search the list of per recipient values of type **SigncryptedException** in the **signcryptedExceptions** component of type **SigncryptedExceptionData** to locate the recipient public-private key pair in the **sids** component of type **SigncryptedException**;
- b) designcrypt the value in the **signcryptionValue** component of type **SigncryptedException** for the recipient using the signcryption algorithm and any associated parameters provided in the **signcryptedExceptionAlgorithm** component of type **SigncryptedException** to recover a value of type **ToBeSigncryptedException**;
- c) perform certificate path validation to gain assurance that the sender public key certificate is trusted.

### 7.2.3 The signcryptedExceptionComponents mode

In signcryptedException-components mode, elements of content of any type or format are signcryptedException for one or more message recipients using the public-private keys of the sender and the public key of each recipient. A content-type specific manifest of signcryptedException element locations in the content is bound to the partially signcryptedException content under a digital signature.

The signcryptedException-components mode is indicated in a message by the following information object identifier:

**signcryptedExceptionComponents MODE ::= { ID signcryptedException-components }**

A value of type **SigncryptedExceptionPartsManifest** is defined in terms of the **&id** and **&Type** fields of the **SIGNCRYPTED** information object class. This type definition uses a parametrized type, **SigncryptedException{}**, whose sole parameter is the **Manifest** information object set:

**SigncryptedExceptionPartsManifest ::= SigncryptedException{{Manifest}}**

```
SigncryptedException{SIGNCRYPTED:IOSet} ::= SEQUENCE {
 Name SIGNCRYPTED.&id({IOSet}),
 Parts SIGNCRYPTED.&Type({IOSet}){@name} OPTIONAL
}
```

**SIGNCRYPTED ::= CLASS {**

```

 &id OBJECT IDENTIFIER UNIQUE,
 &Type OPTIONAL
 }
 WITH SYNTAX {OID &id [PARMS &Type]}

 Manifest SIGNCRYPTED ::= {
 xPathManifest,
 ...
 }

```

The type of manifest required to identify the signcrypt components of a particular object varies with the type, structure and format of the object. For content in the form of an image, a **signcrypt** components manifest might be composed of a list of ((x, y), (x, y)) coordinate pairs that define a rectangular area of the image to be signcrypt and thereby redacted.

In this Recommendation | International Standard, a single manifest object is defined to support component identification in an extensible markup language (XML) document. The **xPathManifest** object uses location paths based on the XPath query language to specify a set of one or more signcrypt XML document components and is defined as follows:

```

xPathManifest SIGNCRYPTED ::= {
 OID id-cms-XPath PARMS XPathSet
}

XPathSet ::= SEQUENCE SIZE(1..MAX) OF XPath

XPath ::= UTF8String (CONSTRAINED BY { -- XML Path Query Language 2.0 -- })

```

A value of type **SigncryptData** using signcrypt-components mode is created as follows:

- a) the value of the **version** component of type **SigncryptData** is set to 1;
- b) in the **contentInformation** component of type **SigncryptData**, set the value of **mode** to the signcrypt-components object identifier. the optional **content** component should not be present;
- c) optionally, include values in the **certificates** component of type **SigncryptData**;
- d) optionally, include values in the **crls** component of type **SigncryptData**;
- e) for each message recipient, include a value of type **Signcrypter** in the **signcrypters** component of type **SigncryptData** as follows:
  - i) the value of the **version** component of type **Signcrypter** is set to 1,
  - ii) in the **sids** component of type **Signcrypter** set the components of a value of type **SigncrypterIDs** to identify the public-private key pairs of the message sender and recipient,
  - iii) signcrypt one or more elements of the content and include the partially signcrypt results as the value of the **signcryptValue** component of type **Signcrypter**
  - iv) in the **signatureInformation** component of type **Signcrypter**, identify the signing key in the optional **signerIdentifier** component if this key differs from the key used to signcrypt,
  - v) if the signature algorithm is not a system default or is not indicated in the signcrypt data algorithm ID, identify the signature algorithm object identifier and any associated parameters in the **signatureAlgorithm** component of type **Signcrypter**,
  - vi) prepare a manifest value in the **signcryptPartsManifest** component of type **ToBeSigned** that indicates the location of each signcrypt element in the **signcryptValue** component of type **Signcrypter**,
  - vii) include the **contentType** and **messageDigest** attributes in the **signedAttributes** component of type **ToBeSigned**,
  - viii) place the results of signing a value of type **ToBeSigned** in the **signatureValue** component of type **SignatureInformation**,
  - ix) optionally, include values in the **unsigncryptAttributes** component of type **Signcrypter**.

To recover the plaintext from the **SigncrypteData** message, a message recipient should perform the following steps for each message recipient:

- search the list of per recipient values of type **Signcrypter** in the **signcrypters** component of type **SigncrypteData** to locate the recipient public-private key pair in the **sids** component of type **Signcrypter**;
- in the **signatureInformation** component of type **Signcrypter**, verify the signature in the **signatureValue** component over a value of type **ToBeSigned** using the signcryption algorithm and any associated parameters provided in the **signcrypteDataAlgorithm** component of type **Signcrypter**;
- using the manifest in the **signcryptePartsManifest** component of type **ToBeSigned**, decrypt each of the indicated signcrypte elements listed in the manifest to recover the partially signcrypte components of the content;
- perform certificate path validation to gain assurance that the sender certificates are trusted.

#### 7.2.4 The signcrypteEnvelope mode

In the signcrypte-envelope mode, a fresh symmetric key is used to encrypt content of any type or format, and the resulting ciphertext is placed in the **encryptedContentInfo** component of a value of type **NamedKeyEncryptedData**, a type defined in ANSI X9.73-2017 that may include an optional key name and optional extensions.

```
NamedKeyEncryptedData ::= SEQUENCE {
 Version CMSVersion,
 keyName [0] OCTET STRING OPTIONAL,
 encryptedContentInfo EncryptedContentInfo,
 unprotectedAttrs [1] UnprotectedEncAttributes OPTIONAL
}
```

The **EncryptedContentInfo** type is defined in CMS protocol as:

```
EncryptedContentInfo ::=
 EncryptedContentInfoType { ContentEncryptionAlgorithmIdentifier }

EncryptedContentInfoType { AlgorithmIdentifierType } ::= SEQUENCE {
 contentType CONTENT-TYPE.&id({ContentSet}),
 contentEncryptionAlgorithm AlgorithmIdentifierType,
 encryptedContent [0] IMPLICIT OCTET STRING OPTIONAL }
```

The **encryptedContentInfo** component of **NamedKeyEncryptedData** type also contains the required algorithm object identifier and any associated parameters of the symmetric encryption algorithm used to encrypt the content. This prepared value of type **NamedKeyEncryptedData** is then encoded as an **OCTET STRING** and placed in the **content** component of the **contentInfo** component of type **SigncrypteData**. The **contentType** component of type **EncryptedContentInfo** is set to the information object identifier value **signcrypte-envelope**.

The **SigncrypteData** message contains one value of type **Signcrypter** for each message recipient. The **Signcrypter** type contains a **signcryptionAttributes** component that must contain the symmetric encryption key signcrypte for each message recipient in a **signcrypteEnvelope** attribute.

The **signcrypteEnvelope** attribute is defined as follows:

```
signcrypteEnvelope ATTRIBUTE ::= {
 WITH SYNTAX SigncrypteKey ID signcrypte-envelope
}
```

```
SigncrypteKey ::= OCTET STRING
```

When the **signcrypteEnvelope** attribute is included in the **signcryptionAttributes** component of type **Signcrypter**, both the **messageDigest** and **contentType** attributes shall be present. To complete construction of the message, the **eContent** is concatenated together with all **signcryptionAttributes** and signed by the message sender. The completed message can then be transmitted to all recipients.

On receiving the message, a recipient first verifies the signature on the message. To recover the payload encrypted in the **signcrypteEnvelope** attribute, a message recipient must first decrypt the content in the value of type **EncapsulatedContentInfo** to recover the symmetric key, then use the recovered key to decrypt the payload.

A value of type **SigncryptedException** using signcryptedException-envelope mode is created as follows:

- a) the value of the **version** component of type **SigncryptedException** is set to 1;
- b) in the **contentInformation** component of type **SigncryptedException**, set the value of mode to the signcryptedException-object identifier.
- c) using a fresh symmetric key, encrypt content of any type or format and place the encrypted content in the optional **encryptedContentInfo** component of a value of type **NamedKeyEncryptedData**;
- d) complete preparation of a value of type **NamedKeyEncryptedData**, by optionally providing a name for the symmetric key in the **keyName** component, and identifying the symmetric content encryption algorithm and any parameters in the **encryptedContent** component;
- e) embed the prepared value of type **NamedKeyEncryptedData** in the optional content component in the **contentInformation** component of type **SigncryptedException**;
- f) prepare a **messageDigest** attribute that identifies a message digest algorithm and contains a digest of the **encryptedContent** component of type **NamedKeyEncryptedData**;
- g) prepare a value of type **ToBeSigncryptedException** that includes this **messageDigest** attribute in the **attributes** component, and the symmetric content encryption key (CEK) in the **contents** component, a value of type **ToBeSigncryptedException** that will be signcryptedException for each message recipient;
- h) optionally, include values in the **certificates** component of type **SigncryptedException**;
- i) optionally, include values in the **crls** component of type **SigncryptedException**;
- j) for each message recipient, include a value of type **SigncryptedException** in the **signcryptedExceptions** component of type **SigncryptedException** as follows:
  - i) the value of the **version** component of type **SigncryptedException** is set to 1,
  - ii) in the **sids** component of type **SigncryptedException** set the components of a value of type **SigncryptedExceptionIDs** to identify the public-private key pairs of the message sender and recipient,
  - iii) set the value of the **signcryptedExceptionAlgorithm** component of type **SigncryptedException** to identify the signcryptedException algorithm and any associated parameters,
  - iv) include the results of signcryptedException the prepared value of type **ToBeSigncryptedException** in the **signcryptedExceptionValue** component of type **SigncryptedException**,
  - v) the optional **signatureInformation** component of type **SigncryptedException** should not be present,
  - vi) Optionally, include values in the **unsigncryptedExceptionAttributes** component of type **SigncryptedException**.

To recover the plaintext from the **SigncryptedException** message, a message recipient should perform the following steps:

- a) search the list of per recipient values of type **SigncryptedException** in the **signcryptedExceptions** component of type **SigncryptedException** to locate the recipient public-private key pair in the **sids** component of type **SigncryptedException**;
- b) decode the optional **content** component in the **contentInformation** component of type **SigncryptedException**, to recover a value of type **NamedKeyEncryptedData** from the octet string;
- c) designcrypt the value in the **signcryptedExceptionValue** component of type **SigncryptedException** for a given recipient using their public-private key pair of the recipient and the public key of the sender to recover the signcryptedException value of type **ToBeSigncryptedException** that contains a symmetric key and a message digest of the content encrypted using the symmetric key;
- d) perform certificate path validation to gain assurance that the sender public key certificate is trusted;
- e) verify that the symmetric key encrypted content has not been modified by computing a digest of the **encryptedContentInfo** component of type **NamedKeyEncryptedData**. using the algorithm in the message digest attribute recovered from the signcryptedException value of type **ToBeSigncryptedException** and comparing the two digests for equivalency;
- f) use the symmetric key recovered from designcryptedException the value of type **ToBeSigncryptedException** to decrypt the content value of the **contentInformation** component of type **SigncryptedException**.

### 7.3 The Signcrypter type

```

Signcrypter ::= SEQUENCE {
 Version Version,
 Sids SigncrypterIDs,
 signcryptDataAlgorithm SigncryptDataAlgorithmIdentifier,
 signcryptValue SigncryptValue,
 signatureInformation SignatureInformation OPTIONAL,
 unsigncryptAttributes UnsigncryptAttributes OPTIONAL
}

```

The public-private key pairs of the message sender and recipient are identified as they are in the **SigncryptData** message as values of type **SigncrypterIDs** defined as follows:

```

SigncrypterIDs ::= SEQUENCE {
 Sender KeyPairIdentifier,
 Recipient KeyPairIdentifier
}

```

```

KeyPairIdentifier ::= SignerIdentifier

```

A message sender uses their own public and private keys along with the public key of the recipient to signcrypt content. The message recipient uses the public key of the sender with their own public and private keys to designcrypt the content signcrypt by the sender.

The **SigncryptDataAlgorithmIdentifier** type contains the signcrypt algorithm ID and any associated parameters. For all of the signcrypt algorithms specified in ISO/IEC 29150, the parameters include one of the hash functions and one of the key derivation functions (KDFs) specified in ISO/IEC 18033-2.

```

SigncryptDataAlgorithmIdentifier ::=
 AlgorithmIdentifier {{SigncryptAlgorithms}}

```

```

SigncryptAlgorithms ALGORITHM ::= {
 SigncryptMechanism, -- ISO/IEC 29150 Signcrypt algorithms --
 ... -- Expect additional algorithm objects --
}

```

A **signcryptValue** component of type **Signcrypter** contains the results of signcrypt, and is defined as follows:

```

SigncryptValue ::= OCTET STRING (SIZE(1..MAX))

```

An optional **signatureInformation** component of type **Signcrypter** contains information needed to support the use of a digital signature when the signcrypt-components mode of processing is used. The definition of type **signatureInformation** is provided in the detailed processing description that follows.

```

SignatureInformation ::= SEQUENCE {
 signerIdentifier SignerIdentifier OPTIONAL,
 signatureAlgorithm SignatureAlgorithmIdentifier OPTIONAL,
 toBeSigned ToBeSigned,
 signatureValue SignatureValue
}

```

```

ToBeSigned ::= SEQUENCE {
 signcryptPartsManifest SigncryptPartsManifest,
 signedAttributes SignedAttributes
}

```

```

SigncryptPartsManifest ::= Signcrypt{{Manifest}}

```

```

Manifest SIGNCRYPTED ::= {
 XPathManifest,
 ... -- Expect additional manifest types --
}

```

```

XPathManifest SIGNCRYPTED ::= {
 OID XPath PARMS XPathSet
}

```

```

XPathSet ::= SEQUENCE (SIZE(1..MAX)) OF XPath

```

**XPath** ::= UTF8String(CONSTRAINED BY { -- XML Path Language 2.0 --})

An optional **unsigncryptAttributes** component of type **Signcrypter** contains any attribute values that may be defined by or required by the message sender. These attributes are not protected by signcrypt data processing. Type **UnsigncryptAttributes** is defined as follows:

```
UnsigncryptAttributes ::=
 SEQUENCE SIZE(1..MAX) OF Attribute {{UnsigncryptAttributes}}

UnsigncryptAttributes ATTRIBUTE ::= {
 ... -- Expect additional attributes --
}
```

## 8 Quantum safe SignedData signatures

Anticipated attacks on current digital signature algorithms using quantum computers will subject documents signed today and requiring long-term protection to increasing security risk over time. This problem is not limited solely to quantum computing risks, such as risk of repudiation by the signer. Long-term signed documents such as a 30 year mortgage are subject to similar risks.

These risks arise as new attacks on signature algorithms are discovered and key length requirements grow with computing power advances. An increase in the number of attacks, the continuing rise in legal and regulatory risks, and changes to the security policies of organizations all add to these risks. It is possible to mitigate some of these security risks using a quantum-safe countersignature over signed content created using current digital signature algorithms.

A countersignature that relies on a quantum-safe signature can be implemented in a **SignedData** message using an optional signed attribute. The schema of a **SignedData** message supports a series of signers represented in a value of type **SignerInfos**. Each signer is represented in this series as a value of type **SignerInfo**.

Type **SignerInfo** is defined as:

```
SignerInfo ::= SEQUENCE {
 version CMSVersion,
 sid SignerIdentifier,
 digestAlgorithm DigestAlgorithmIdentifier,
 signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
 signatureAlgorithm SignatureAlgorithmIdentifier,
 signature SignatureValue,
 unsignedAttrs [1] IMPLICIT Attributes{{UnsignedAttributes}}
 OPTIONAL }
```

Type **SignerInfo** allows each signer to use a different signing key, message digest, and signature algorithm. Each signer can include their own set of attributes that will be cryptographically bound under their signature. A **signerInfos** attribute collects this series of values into an attribute that can be included in the signed attributes of a counter-signer.

The **signerInfos** attribute is as follows:

```
aa-signerInfos ATTRIBUTE ::=
 {TYPE SignerInfos IDENTIFIED BY id-signerInfos}
```

```
id-signerInfos OBJECT IDENTIFIER ::= {itu-t recommendation(0) x(24)
 cms-profile(894) attribute(2) signerInfos(1)}
```

The **signerInfos** attribute contains a value of type **SignerInfos**, a series of values of type **SignerInfo**, one value for each signer of the **SignedData** content. Each cosigner shall sign content using their choice of signature and message digest algorithm. The signing key of each cosigner shall be included in the **SignerInfo** value of the cosigner using any of the choice alternatives defined in the **SignerIdentifier** type.

The optional **signedAttrs** component of type **SignerInfo** shall be present in the message. At a minimum, each cosigner shall include a **contentType** attribute and a **messageDigest** attribute in the **signedAttrs** component of their **SignerInfo** value. Additional signed attributes of any type or format may also be included by each cosigner. Any number or type of unsigned attributes may also be included by each cosigner in the **unsignedAttrs** component of their **SignerInfo** value.

The values of type **SignerInfo** created by all signers shall follow other defined processing. During signature verification of a **SignedData** message, a relying party may treat the **signerInfos** attribute as an opaque string. Applications that recognize this attribute may choose to defer signature verification processing. Failure of one or more cosigner **SignerInfo** values shall be handled as defined by the application.

## 8.1 Detached content consideration

When the detached form of **SignedData** is used, the content of the message is not present in the **SignedData** type. This message content must be available during signing and signature verification operations so that a message digest of the signed content can be calculated.

When the default content location is not known to the communicating parties, content signers can include a **contentLocation** attribute in their signed attributes. This attribute can also be used when it is necessary for a consigner to indicate a different detached object, such as a language-specific version of a contract.

The **contentLocation** attribute is defined as follows:

```
aa-contentLocation ATTRIBUTE ::=
 {TYPE URI IDENTIFIED BY id-contentLocation}

URI ::= UTF8String(SIZE(1..MAX))

id-contentLocation OBJECT IDENTIFIER ::= {itu-t recommendation(0) x(24)
 cms-profile(894) attribute(2) contentLocation(2)}
```

A value of type **URI** is a uniform resource identifier (URI) value that points to a location of detached **SignedData** content. A **contentLocation** attribute can be included in a **SignedAttributes** component of a **SignerInfo** component of type **SignedData** of any signer of the message. In some applications, it may be convenient to include a single content location attribute in the signed attributes of the countersigner.

## 8.2 Time stamp consideration

Time stamps included in **SignedData** can be used to demonstrate that the validity period of a signer certificate included the time of signing a message. Long-term signatures may need to be verified after the validity period of a signing certificate has expired. A time stamp attribute that is included in the **SignedAttributes** component can be compared by a relying party to the validity period of the signer certificate to ensure the certificate was valid for use when the message was signed.

The **timeStamped** attribute is defined as follows:

```
aa-timeStamped ATTRIBUTE ::=
 {TYPE TimeStamped IDENTIFIED BY id-timeStamped}

TimeStamped ::= SEQUENCE {
 timeStampValue TimeStamp,
 timeStampService URI OPTIONAL
}

TimeStamp ::= CHOICE {
 timeStampToken TimeStampToken,
 localTimeStamp GeneralizedTime,
 ... -- Expect additional time types --
}

id-timeStamped OBJECT IDENTIFIER ::= {itu-t recommendation(0) x(24)
 cms-profile(894) attribute(2) timeStamped(5)}
```

The type **TimeStamped** contains two components, a required **timeStampValue** and an optional **timeStampService** that indicates the location of a time stamp service provider that can validate the time stamp. A **timeStampValue** component is a value of type **TimeStamp**, a choice between two alternatives, an ANSI X9.95 trusted timestamp token or a value from a local time source.

```
TimeStamp ::= CHOICE {
 TimeStampToken TimeStampToken, -- X9.95 Trusted Time Stamp --
 localTimeStamp GeneralizedTime
}
```

The first choice alternative of type **TimeStamp** may be any of the four types of tokens defined in ANSI X9.95.

### 8.3 The tokenizedParts attribute

The **tokenizedParts** attribute carries a manifest, a list of the locations of elements within content of any type or format that has been tokenized. Tokenization techniques provide confidentiality services for the elements of the content specified in the manifest. A manifest can be constructed on a per-message recipient basis to achieve granular message redaction. The format and the information contained in the manifest varies with the type of content and user requirements. Only one manifest is defined in this Recommendation | International Standard, a simple manifest for XML content, though the set of possible manifests is extensible and unbounded.

XML Path (XPath) expressions can be used to locate any tokenized element in any XML-instance document. To identify the tokenization components in an XML-instance document, a set of XPath expressions can be used to identify the location of each tokenized element. When the contents of an XML element are tokenized, the sender includes the outer markup tags in the tokenization operation.

These outer tags are not removed from the document, since they are used to locate the element using XPath and compared for assurance to the bounding tags in a detokenized value. The markup between these outer tags is replaced with a character string representation of the tokenization results, a value of XML type `base64Binary`. This XML type is used to represent arbitrary Base64-encoded binary data in a human-readable, 'mail-safe' format. A message recipient uses the list of XPath expressions to locate the tags in an XML-instance document that contain tokenized data.

The entire document containing tokenized components and the associated manifest attribute are signed to cryptographically bind them under a digital signature. The signature prevents any part of the document or attributes from being substituted or modified without detection.

Successful signature verification provides a relying party assurance of the authenticity of the document and the manifest attribute. A tokenization manifest attribute is protected under the signature and indicates that the tokenization components of the document have integrity and confidentiality. The signature on the partially tokenized content can then be verified and its plaintext content recovered. The recovered plaintext can then be used to replace the tokenized value and surrounding tags with the recovered XML markup prior to performing XML schema validation

## 9 Other key management techniques

### 9.1 Constructive key management

Constructive key management (CKM) is a dynamic symmetric key management framework described in ANSI X9.69-2017. It is a method of establishing a key, whereby several components of keying material, both symmetric and asymmetric type of keys, where each component is used for a specific purpose, are combined using a mathematical function to produce an object key (content key).

Internal keys associated with CKM meet criteria established in ISO 11568-1. The mathematical function inherent in CKM is a two-part process for binding an analogue representation like a naming convention to an asymmetric or symmetric type of key and having that part of the process used to provide integrity to a second part that includes a random number and other symmetric key functionality, which together results in a CEK. The first part of the process establishes labels (or attributes) that define characteristics of protected data through a naming convention to create a cryptographic enforced access control for content (as objects) resulting from the encryption action related to the CEK. Protecting the data and access control for the data is enforced by the framework. Data peerage can be done through combinations of logic relationships. Each encryption event results in a new random number. A key recovery capability is inherent in the framework and an administrative capability identified in ANSI X9.69-2017.

```
ckmRecipientInfo KEY-MANAGEMENT ::=
 { KeyConstructRecipientInfo IDENTIFIED BY id-ckm-recipient-info }
```

is the information object that is used to populate the components of type **OtherRecipientInfo**. The object identifier value in component **oriType** is the value **id-ckm-recipient-info**, and the associated type is a value in component **oriValue** of type **KeyConstructRecipientInfo** in its encoded format.

In an XML encoding, a CKM value of type **OtherRecipientInfo** is

```
<OtherRecipientInfo>
 <oriType>
 1.2.840.10060.2.1 <!-- id-ckm-recipient-info -->
 </oriType>
 <oriValue>
 <KeyConstructRecipientInfo>
```

```

<version> 1 </version>
<did>
 <domainName>domainName</domainName>
 <domainMaintenanceLevel>1</domainMaintenanceLevel>
</did>
<ckmid>
 <unencrypted> 1 2 3</unencrypted>
</ckmid>
<ukm>some-base64-text</ukm>
<keyConstructionAlgorithm>
 <algorithm>algorithmIdentifierOid</algorithm>
</keyConstructionAlgorithm>
<encryptedRandom>some-base64-text</encryptedRandom>
</KeyConstructRecipientInfo>
</oriValue>
<OtherRecipientInfo>

```

In this example, **ukm** and **encryptedRandom** values contain octet string values encoded in base 64. The **algorithmIdentifier** value contains an object identifier value.

## 9.2 Database encryption key management

Database encryption key management (DBEKM) is a key management technique defined in ANSI X9.73-2017 for managing symmetric keys in a database environment. This technique relies on the secure exchange of information between a secure cryptographic device (SCD) and a server. The **dbekmRecipientInfo** object of class **KEY-MANAGEMENT** is defined as follows:

```

dbekmRecipientInfo KEY-MANAGEMENT ::=
 { DBEKMRecipientInfo IDENTIFIED BY id-dbekm-recv-info }

```

This information object that is used to populate the components of type **OtherRecipientInfo**. The object identifier value in component **oriType** is the value **id-dbekm-recv-info**, and the associated type is a value in component **oriValue** of type **DBEKMRecipientInfo** in its encoded format.

The DBEKM method is designed to support using a data encryption key (DK) in cryptographic software and using a master key in cryptographic hardware, also referred to as an SCD without violating the cryptographic boundary of the SCD or spoofing the SCD by presenting the SCD with a data object that is intended for use as a cryptographic key. Further, DBEKM eliminates the need of relying on a software object using a password-based key derivation function (PBKDF) to protect data encryption keys.

The DBEKM method employs three key types:

- 1) master key encryption key (MK) used to protect the hashed message authentication (HMAC) key (HK);
- 2) keyed HK used to generate a seed;
- 3) DK used to protect data.

The DBEKM method employs four cryptographic functions:

- 1) algorithms for data and key encryption, i.e., AES;
- 2) algorithms for keyed HMAC;
- 3) algorithms for hashing, e.g. secure hash algorithm 2 (SHA-2), secure hash algorithm 3 (SHA-3);
- 4) algorithms for KDFs, i.e. ISO/IEC 11770-6.

The DBEKM method has the following benefits:

- 1) the cryptographic hardware generates the MK and the HK;
- 2) the cryptographic hardware retains the MK, but not the HK;
- 3) the cryptographic hardware generates a seed per the unique database ID;
- 4) the cryptographic hardware does not know the KDF and cannot generate the DK;
- 5) the server does not store any cleartext keys or passwords protecting keys;
- 6) the server manages its own unique database ID.

The DBEKM method has the following security considerations:

- 1) the seed is securely transmitted from the cryptographic hardware to the database server;
- 2) the server cryptographic software generates a DK using a KDF based on the seed;
- 3) the server memory contains the cleartext DK for use with cryptographic software.

The DBEKM method can be used for generating a DK as a CEK, for changing a DK, using multiple DKs with a single server when protecting different data elements, and using multiple DKs or MKs with multiple servers.

### 9.2.1 Single server initial data encryption key

For a single server initial DK, a secure connection is first established between the SCD and the server storage. The SCD generates the MK and HK, encrypts the HK using the MK, and sends the HK cryptogram to the server over a secured channel. The server stores the HK cryptogram and at some previous or subsequent point in time generates a unique ID that represents a meaningful database name. Meanwhile, the SCD destroys the HK, but retains the MK.

To obtain a seed for the DK, the server sends a request to the SCD containing the HK cryptogram and the unique ID over a secured channel. The SCD decrypts the HK using the MK, generates a seed using the HMAC algorithm with the HK and the unique ID, and sends the seed to the server over a secured channel. The server generates the DK using a KDF with the seed and installs the DK into memory for data encryption and decryption. Meanwhile, the SCD destroys the HK and the seed.

### 9.2.2 Single server data encryption key change

For a single server DK change, a secure connection is first established between the SCD and the server storage. The server can change the DK at any time by changing the unique ID. The data was previously encrypted by sending a request to the SCD with the HK cryptogram and unique ID, generating the DK using the KDF with the seed received from the SCD, and encrypting the data using the DK in memory.

To change the DK the server decrypts the data, generates and stores a new ID, sends a request to the SCD with the HK cryptogram and the new ID, generates a new DK using the new seed, and encrypts the data using the new DK in memory. The server can decrypt with the old DK and re-encrypt with the new DK all of the data at once, or manage a gradual migration between the two DK.

The SCD only retains the MK, it destroys the old and new DK, the old and new ID, and the old and new seeds. As long as the server can manage its IDs, it can manage its DKs accordingly, but it cannot generate the DK without obtaining the seed from the SCD. The server can change the DK at any time by managing its ID, and if needed can recover old DK as long as it archives the HK cryptogram and the associated ID.

### 9.2.3 Multiple data encryption keys with a single server

A single server might need multiple encryption keys to protect different information on a single server instead of using the same key for all data that needs encryption. The server can manage multiple DKs by managing multiple IDs. The SCD generates a single MK and single HK, encrypts the HK using the MK, and sends the HK cryptogram to the server over a secured channel. The server stores the HK cryptogram, and at some previous or subsequent point in time, the server generates multiple unique IDs, one for each DK. Meanwhile, the SCD destroys the HK, but retains the MK.

To obtain a seed  $s_1$  for the first DK<sub>1</sub>, the server sends a request to the SCD containing the HK cryptogram and the first unique ID<sub>1</sub> over a secured channel. The SCD decrypts the HK using the MK, generates seed  $s_1$  using the HMAC algorithm with the HK and the unique ID<sub>1</sub>, and sends seed  $s_1$  to the server over a secured channel. The server generates the first DK<sub>1</sub> using a KDF with seed  $s_1$  and installs the DK<sub>1</sub> into memory for data encryption and decryption. Meanwhile, the SCD destroys the HK and seed  $s_1$ .

To obtain a seed  $s_2$  for the second DK<sub>2</sub>, the server sends a request to the SCD containing the HK cryptogram and the second unique ID<sub>2</sub> over a secured channel. The SCD decrypts the HK using the MK, generates seed  $s_2$  using the HMAC algorithm with the HK and the unique ID<sub>2</sub>, and sends seed  $s_2$  to the server over a secured channel. The server generates the second DK<sub>2</sub> using a KDF with seed  $s_2$  and installs the DK<sub>2</sub> into memory for data encryption and decryption. Meanwhile, the SCD destroys the HK and seed  $s_2$ . When the server is restarted and the DKs are erased from memory, the server can regenerate the DKs by reacquiring the seeds at any time by resending the same HK cryptogram and unique IDs to the SCD.

#### 9.2.4 Multiple data encryption keys with multiple servers

Multiple servers collocated within the same data-centre might need a unique encryption key per server instead of sharing the same key on all servers. A secure connection is first established between the SCD and each storage of each server. The SCD generates a single MK and single HK, encrypts the HK using the MK, and sends the HK cryptogram to both servers over a secured channel. The servers store the HK cryptogram and at some previous or subsequent point in time, each server generates a unique identifier, ID1 for server S1 and ID2 for server S2. Meanwhile, the SCD destroys the HK, but retains the MK.

To obtain a seed s1 for DK1, the server S1 sends a request to the SCD containing the HK cryptogram and the unique ID1 over a secured channel. The SCD decrypts the HK using the MK, generates seed s1 using the HMAC algorithm with the HK and the unique ID1, and sends seed s1 to server S1 over a secured channel. The server S1 generates DK1 using a KDF with seed s1 and installs the DK1 into memory for data encryption and decryption. Meanwhile, the SCD destroys the HK and seed s1.

To obtain a seed s2 for DK2, the server S2 sends a request to the SCD containing the HK cryptogram and the second unique ID2 over a secured channel. The SCD decrypts the HK using the MK, generates seed s2 using the HMAC algorithm with the HK and the unique ID2, and sends seed s2 to server S2 over a secured channel. The server generates DK2 using a KDF with seed s2 and installs the DK2 into memory for data encryption and decryption. Meanwhile, the SCD destroys the HK and seed s2. When any of the servers are restarted its DK is erased from memory, but the servers can regenerate their DK by reacquiring the seed at any time, by resending the same HK cryptogram and unique ID to the SCD.

#### 9.2.5 Multiple hashed message authentication keys with multiple servers

Multiple servers deployed regionally in different data-centres might need a unique HK per server instead of sharing the same key on all servers. The SCD generates a single MK and multiple HKs, one for each server, encrypts each HK using the MK and sends each HK cryptogram to the corresponding server over a secured channel. The servers store the HK cryptogram and at some previous or subsequent point in time each one generates a unique identifier, ID1 for server S1 and ID2 for server S2. Meanwhile, the SCD destroys the HKs, but retains the MK.

To obtain a seed s1 for DK1, the server S1 sends a request to the SCD containing the HK1 cryptogram and the unique ID1 over a secured channel. The SCD decrypts HK1 using the MK, generates seed s1 using the HMAC algorithm with HK1 and the unique ID1, and sends seed s1 to server S1 over a secured channel. The server S1 generates DK1 using a KDF with seed s1 and installs the DK1 into memory for data encryption and decryption. Meanwhile, the SCD destroys HK1 and seed s1.

To obtain a seed s2 for DK2 the server S2 sends a request to the SCD containing the HK2 cryptogram and the second unique ID2 over a secured channel. The SCD decrypts HK2 using the MK, generates seed s2 using the HMAC algorithm with HK2 and the unique ID2, and sends seed s2 to server S2 over a secured channel. The server generates DK2 using a KDF with seed s2 and installs the DK2 into memory for data encryption and decryption. Meanwhile, the SCD destroys the HK and seed s2. When any of the servers is restarted, its DK is erased from memory, but the servers can regenerate the DK by reacquiring the seed at any time by resending the same HK cryptogram and unique ID to the SCD.

## Annex A

## ASN.1 modules

(This annex forms an integral part of this Recommendation | International Standard.)

## A.1 Main CMS module (from IETF RFC 6268)

```

CryptographicMessageSyntax-2010
 { iso(1) member-body(2) us(840) rsadsi(113549)
 pkcs(1) pkcs-9(9) smime(16) modules(0) id-mod-cms-2009(58) }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

IMPORTS

ParamOptions, DIGEST-ALGORITHM, SIGNATURE-ALGORITHM,
PUBLIC-KEY, KEY-DERIVATION, KEY-WRAP, MAC-ALGORITHM,
KEY-AGREE, KEY-TRANSPORT, CONTENT-ENCRYPTION, ALGORITHM,
AlgorithmIdentifier{
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)}

SignatureAlgs, MessageDigestAlgs, KeyAgreementAlgs,
MessageAuthAlgs, KeyWrapAlgs, ContentEncryptionAlgs,
KeyTransportAlgs, KeyDerivationAlgs, KeyAgreePublicKeys
FROM CryptographicMessageSyntaxAlgorithms-2009
{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) modules(0) id-mod-cmsalg-2001-02(37) }

Certificate, CertificateList, CertificateSerialNumber, Name, ATTRIBUTE
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) }

AttributeCertificate
FROM PKIXAttributeCertificate-2009
{ iso(1) identified-organization(3) dod(6) internet(1)
security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-attribute-cert-02(47) }

AttributeCertificateV1
FROM AttributeCertificateVersion1-2009
{iso(1) identified-organization(3) dod(6) internet(1) security(5)
mechanisms(5) pkix(7) id-mod(0) id-mod-v1AttrCert-02(49)}

CMSProfileAttributes
FROM CMSProfileAttributes
{itu-t recommendation(0) x(24) cms-profile(894) module(0)
CMSProfileAttributes(3) version1(1)}

tokenizedParts
FROM TokenizationManifest
{iso(1) identified-organization(3) tc68(133) country(16)
x9(840) x9Standards(9) x9-73(73) module(0) tokenMan(7)};

-- Cryptographic Message Syntax
-- The following are used for version numbers using the ASN.1
-- NOTE: The document reference represents where the versioned
-- feature was introduced to the module.
--
-- idiom "[[n:"
-- Version 1 = PKCS #7
-- Version 2 = S/MIME V2
-- Version 3 = RFC 2630
-- Version 4 = RFC 3369
-- Version 5 = RFC 3852

CONTENT-TYPE ::= CLASS {
 &id OBJECT IDENTIFIER UNIQUE,

```

```

 &Type OPTIONAL
 } WITH SYNTAX {
 [TYPE &Type] IDENTIFIED BY &id
 }

ContentType ::= CONTENT-TYPE.&id

ContentInfo ::= SEQUENCE {
 contentType CONTENT-TYPE.&id({ContentSet}),
 content [0] EXPLICIT CONTENT-TYPE.&Type({ContentSet}{@contentType})
}

ContentSet CONTENT-TYPE ::= {
 -- Define the set of content types to be recognized.
 ct-Data | ct-SignedData | ct-EncryptedData | ct-EnvelopedData |
 ct-AuthenticatedData | ct-DigestedData, ... }

SignedData ::= SEQUENCE {
 version CMSVersion,
 digestAlgorithms SET OF DigestAlgorithmIdentifier,
 encapContentInfo EncapsulatedContentInfo,
 certificates [0] IMPLICIT CertificateSet OPTIONAL,
 crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,
 signerInfos SignerInfos }

SignerInfos ::= SET OF SignerInfo

EncapsulatedContentInfo ::= SEQUENCE {
 eContentType CONTENT-TYPE.&id({ContentSet}),
 eContent [0] EXPLICIT OCTET STRING
 (CONTAINING CONTENT-TYPE.&Type({ContentSet}{@eContentType})) OPTIONAL }

SignerInfo ::= SEQUENCE {
 version CMSVersion,
 sid SignerIdentifier,
 digestAlgorithm DigestAlgorithmIdentifier,
 signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
 signatureAlgorithm SignatureAlgorithmIdentifier,
 signature SignatureValue,
 unsignedAttrs [1] IMPLICIT Attributes({UnsignedAttributes}) OPTIONAL }

SignedAttributes ::= Attributes {{ SignedAttributesSet }}

SignerIdentifier ::= CHOICE {
 issuerAndSerialNumber IssuerAndSerialNumber,
 ...,
 [[3: subjectKeyIdentifier [0] SubjectKeyIdentifier]] }

SignedAttributesSet ATTRIBUTE ::=
 { aa-signingTime | aa-messageDigest | aa-contentType, |
 CMSProfileAttributes, tokenizedParts ... }

UnsignedAttributes ATTRIBUTE ::= { aa-countersignature, ... }

SignatureValue ::= OCTET STRING

EnvelopedData ::= SEQUENCE {
 version CMSVersion,
 originatorInfo [0] IMPLICIT OriginatorInfo OPTIONAL,
 recipientInfos RecipientInfos,
 encryptedContentInfo EncryptedContentInfo,
 ...,
 [[2: unprotectedAttrs [1] IMPLICIT Attributes
 {{ UnprotectedEnvAttributes }} OPTIONAL]] }

OriginatorInfo ::= SEQUENCE {
 certs [0] IMPLICIT CertificateSet OPTIONAL,
 crls [1] IMPLICIT RevocationInfoChoices OPTIONAL }

RecipientInfos ::= SET SIZE (1..MAX) OF RecipientInfo

EncryptedContentInfo ::=
 EncryptedContentInfoType { ContentEncryptionAlgorithmIdentifier }

EncryptedContentInfoType { AlgorithmIdentifierType } ::= SEQUENCE {
 contentType CONTENT-TYPE.&id({ContentSet}),

```

```

 contentEncryptionAlgorithm AlgorithmIdentifierType,
 encryptedContent [0] IMPLICIT OCTET STRING OPTIONAL }

-- If you want to do constraints, you might use:
-- EncryptedContentInfo ::= SEQUENCE {
-- contentType CONTENT-TYPE.&id({ContentSet}),
-- contentEncryptionAlgorithm ContentEncryptionAlgorithmIdentifier,
-- encryptedContent [0] IMPLICIT ENCRYPTED {CONTENT-TYPE.
-- &Type({ContentSet}){@contentType}} OPTIONAL }

-- ENCRYPTED {ToBeEncrypted} ::= OCTET STRING (CONSTRAINED BY { ToBeEncrypted })

UnprotectedEnvAttributes ATTRIBUTE ::= { ... }
UnprotectedEncAttributes ATTRIBUTE ::= { ... }

RecipientInfo ::= CHOICE {
 ktri KeyTransRecipientInfo,
 ...,
 [[3: kari [1] KeyAgreeRecipientInfo]],
 [[4: kekri [2] KEKRecipientInfo]],
 [[5: pwri [3] PasswordRecipientInfo,
 ori [4] OtherRecipientInfo]] }

EncryptedKey ::= OCTET STRING

KeyTransRecipientInfo ::= SEQUENCE {
 version CMSVersion, -- always set to 0 or 2
 rid RecipientIdentifier,
 keyEncryptionAlgorithm AlgorithmIdentifier
 {KEY-TRANSPORT, {KeyTransportAlgorithmSet}},
 encryptedKey EncryptedKey }

KeyTransportAlgorithmSet KEY-TRANSPORT ::= { KeyTransportAlgs, ... }

RecipientIdentifier ::= CHOICE {
 issuerAndSerialNumber IssuerAndSerialNumber,
 ...,
 [[2: subjectKeyIdentifier [0] SubjectKeyIdentifier]] }

KeyAgreeRecipientInfo ::= SEQUENCE {
 version CMSVersion, -- always set to 3
 originator [0] EXPLICIT OriginatorIdentifierOrKey,
 ukm [1] EXPLICIT UserKeyingMaterial OPTIONAL,
 keyEncryptionAlgorithm AlgorithmIdentifier
 {KEY-AGREE, {KeyAgreementAlgorithmSet}},
 recipientEncryptedKeys RecipientEncryptedKeys }

KeyAgreementAlgorithmSet KEY-AGREE ::= { KeyAgreementAlgs, ... }

OriginatorIdentifierOrKey ::= CHOICE {
 issuerAndSerialNumber IssuerAndSerialNumber,
 subjectKeyIdentifier [0] SubjectKeyIdentifier,
 originatorKey [1] OriginatorPublicKey }

OriginatorPublicKey ::= SEQUENCE {
 algorithm AlgorithmIdentifier {PUBLIC-KEY, {OriginatorKeySet}},
 publicKey BIT STRING }

OriginatorKeySet PUBLIC-KEY ::= { KeyAgreePublicKeys, ... }

RecipientEncryptedKeys ::= SEQUENCE OF RecipientEncryptedKey

RecipientEncryptedKey ::= SEQUENCE {
 rid KeyAgreeRecipientIdentifier,
 encryptedKey EncryptedKey }

KeyAgreeRecipientIdentifier ::= CHOICE {
 issuerAndSerialNumber IssuerAndSerialNumber,
 rKeyId [0] IMPLICIT RecipientKeyIdentifier }

RecipientKeyIdentifier ::= SEQUENCE {
 subjectKeyIdentifier SubjectKeyIdentifier,
 date GeneralizedTime OPTIONAL,
 other OtherKeyAttribute OPTIONAL }

```

```

SubjectKeyIdentifier ::= OCTET STRING

KEKRecipientInfo ::= SEQUENCE {
 version CMSVersion, -- always set to 4
 kekid KEKIdentifier,
 keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
 encryptedKey EncryptedKey }

KEKIdentifier ::= SEQUENCE {
 keyIdentifier OCTET STRING,
 date GeneralizedTime OPTIONAL,
 other OtherKeyAttribute OPTIONAL }

PasswordRecipientInfo ::= SEQUENCE {
 version CMSVersion, -- always set to 0
 keyDerivationAlgorithm [0] KeyDerivationAlgorithmIdentifier
 OPTIONAL,
 keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
 encryptedKey EncryptedKey }

OTHER-RECIPIENT ::= TYPE-IDENTIFIER

OtherRecipientInfo ::= SEQUENCE {
 oriType OTHER-RECIPIENT.&id({SupportedOtherRecipInfo}),
 oriValue OTHER-RECIPIENT.&Type({SupportedOtherRecipInfo}{@oriType})}

SupportedOtherRecipInfo OTHER-RECIPIENT ::= { ... }

DigestedData ::= SEQUENCE {
 version CMSVersion,
 digestAlgorithm DigestAlgorithmIdentifier,
 encapContentInfo EncapsulatedContentInfo,
 digest Digest, ... }

Digest ::= OCTET STRING

EncryptedData ::= SEQUENCE {
 version CMSVersion,
 encryptedContentInfo EncryptedContentInfo,
 ...,
 [[2: unprotectedAttrs [1] IMPLICIT Attributes
 {{UnprotectedEncAttributes}} OPTIONAL]] }

AuthenticatedData ::= SEQUENCE {
 version CMSVersion,
 originatorInfo [0] IMPLICIT OriginatorInfo OPTIONAL,
 recipientInfos RecipientInfos,
 macAlgorithm MessageAuthenticationCodeAlgorithm,
 digestAlgorithm [1] DigestAlgorithmIdentifier OPTIONAL,
 encapContentInfo EncapsulatedContentInfo,
 authAttrs [2] IMPLICIT AuthAttributes OPTIONAL,
 mac MessageAuthenticationCode,
 unauthAttrs [3] IMPLICIT UnauthAttributes OPTIONAL }

AuthAttributes ::= SET SIZE (1..MAX) OF Attribute {{AuthAttributeSet}}

AuthAttributeSet ATTRIBUTE ::= { aa-contentType | aa-messageDigest | aa-signingTime, ...}

MessageAuthenticationCode ::= OCTET STRING

UnauthAttributes ::= SET SIZE (1..MAX) OF Attribute {{UnauthAttributeSet}}

UnauthAttributeSet ATTRIBUTE ::= {...}

--
-- General algorithm definitions
--

DigestAlgorithmIdentifier ::= AlgorithmIdentifier
 {DIGEST-ALGORITHM, {DigestAlgorithmSet}}

DigestAlgorithmSet DIGEST-ALGORITHM ::= {
 CryptographicMessageSyntaxAlgorithms-2009.MessageDigestAlgs, ... }

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier
 {SIGNATURE-ALGORITHM, {SignatureAlgorithmSet}}

```

```

SignatureAlgorithmSet SIGNATURE-ALGORITHM ::= { SignatureAlgs, ... }

KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
 { KEY-WRAP, { KeyEncryptionAlgorithmSet } }

KeyEncryptionAlgorithmSet KEY-WRAP ::= { KeyWrapAlgs, ... }

ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
 { CONTENT-ENCRYPTION, { ContentEncryptionAlgorithmSet } }

ContentEncryptionAlgorithmSet CONTENT-ENCRYPTION ::=
 { ContentEncryptionAlgs, ... }

MessageAuthenticationCodeAlgorithm ::= AlgorithmIdentifier
 { MAC-ALGORITHM, { MessageAuthenticationCodeAlgorithmSet } }

MessageAuthenticationCodeAlgorithmSet MAC-ALGORITHM ::= { MessageAuthAlgs, ... }

KeyDerivationAlgorithmIdentifier ::= AlgorithmIdentifier
 { KEY-DERIVATION, { KeyDerivationAlgs, ... } }

RevocationInfoChoices ::= SET OF RevocationInfoChoice

RevocationInfoChoice ::= CHOICE {
 crl CertificateList,
 ...,
 [[5: other [1] IMPLICIT OtherRevocationInfoFormat]] }

OTHER-REVOK-INFO ::= TYPE-IDENTIFIER

OtherRevocationInfoFormat ::= SEQUENCE {
 otherRevInfoFormat OTHER-REVOK-INFO.&id({SupportedOtherRevokInfo}),
 otherRevInfo OTHER-REVOK-INFO.&Type
 ({SupportedOtherRevokInfo}{@otherRevInfoFormat}) }

SupportedOtherRevokInfo OTHER-REVOK-INFO ::= { ... }

CertificateChoices ::= CHOICE {
 certificate Certificate,
 extendedCertificate [0] IMPLICIT ExtendedCertificate,
 -- Obsolete
 ...,
 [[3: v1AttrCert [1] IMPLICIT AttributeCertificateV1]],
 -- Obsolete
 [[4: v2AttrCert [2] IMPLICIT AttributeCertificateV2]],
 [[5: other [3] IMPLICIT OtherCertificateFormat]] }

AttributeCertificateV2 ::= AttributeCertificate

OTHER-CERT-FMT ::= TYPE-IDENTIFIER

OtherCertificateFormat ::= SEQUENCE {
 otherCertFormat OTHER-CERT-FMT.&id({SupportedCertFormats}),
 otherCert OTHER-CERT-FMT.&Type
 ({SupportedCertFormats}{@otherCertFormat}) }

SupportedCertFormats OTHER-CERT-FMT ::= { ... }

CertificateSet ::= SET OF CertificateChoices

IssuerAndSerialNumber ::= SEQUENCE {
 issuer Name,
 serialNumber CertificateSerialNumber }

CMSVersion ::= INTEGER { v0(0), v1(1), v2(2), v3(3), v4(4), v5(5) }

UserKeyingMaterial ::= OCTET STRING

KEY-ATTRIBUTE ::= TYPE-IDENTIFIER

OtherKeyAttribute ::= SEQUENCE {
 keyAttrId KEY-ATTRIBUTE.&id({SupportedKeyAttributes}),
 keyAttr KEY-ATTRIBUTE.&Type({SupportedKeyAttributes}{@keyAttrId}) }

SupportedKeyAttributes KEY-ATTRIBUTE ::= { ... }

-- Content Type Object Identifiers

```

```

id-ct-contentInfo OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) ct(1) 6 }

ct-Data CONTENT-TYPE ::= { IDENTIFIED BY id-data }

id-data OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) rsadsi(113549) pkcs(1) pkcs7(7) 1 }

ct-SignedData CONTENT-TYPE ::=
 { TYPE SignedData IDENTIFIED BY id-signedData }

id-signedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2 }

ct-EnvelopedData CONTENT-TYPE ::=
 { TYPE EnvelopedData IDENTIFIED BY id-envelopedData }

id-envelopedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) rsadsi(113549) pkcs(1) pkcs7(7) 3 }

ct-DigestedData CONTENT-TYPE ::=
 { TYPE DigestedData IDENTIFIED BY id-digestedData }

id-digestedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) rsadsi(113549) pkcs(1) pkcs7(7) 5 }

ct-EncryptedData CONTENT-TYPE ::=
 { TYPE EncryptedData IDENTIFIED BY id-encryptedData }

id-encryptedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) rsadsi(113549) pkcs(1) pkcs7(7) 6 }

ct-AuthenticatedData CONTENT-TYPE ::=
 { TYPE AuthenticatedData IDENTIFIED BY id-ct-authData }

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

--
-- The CMS Attributes
--

MessageDigest ::= OCTET STRING

SigningTime ::= Time

Time ::= CHOICE {
 utcTime UTCTime,
 generalTime GeneralizedTime }

Countersignature ::= SignerInfo

-- Attribute and object Identifiers

aa-contentType ATTRIBUTE ::=
 { TYPE ContentType IDENTIFIED BY id-contentType }

id-contentType OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) rsadsi(113549) pkcs(1) pkcs9(9) 3 }

aa-messageDigest ATTRIBUTE ::=
 { TYPE MessageDigest IDENTIFIED BY id-messageDigest }

id-messageDigest OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) rsadsi(113549) pkcs(1) pkcs9(9) 4 }

aa-signingTime ATTRIBUTE ::=
 { TYPE SigningTime IDENTIFIED BY id-signingTime }

id-signingTime OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) rsadsi(113549) pkcs(1) pkcs9(9) 5 }

aa-countersignature ATTRIBUTE ::=
 { TYPE Countersignature IDENTIFIED BY id-countersignature }

id-countersignature OBJECT IDENTIFIER ::= { iso(1) member-body(2)
 us(840) rsadsi(113549) pkcs(1) pkcs9(9) 6 }

--

```

```

-- Obsolete Extended Certificate syntax from PKCS#6
--

ExtendedCertificateOrCertificate ::= CHOICE {
 certificate Certificate,
 extendedCertificate [0] IMPLICIT ExtendedCertificate }

ExtendedCertificate ::= SEQUENCE {
 extendedCertificateInfo ExtendedCertificateInfo,
 signatureAlgorithm SignatureAlgorithmIdentifier,
 signature Signature }

ExtendedCertificateInfo ::= SEQUENCE {
 version CMSVersion,
 certificate Certificate,
 attributes UnauthAttributes }

Signature ::= BIT STRING

Attribute{ ATTRIBUTE:AttrList } ::= SEQUENCE {
 attrType ATTRIBUTE.&id({AttrList}),
 attrValues SET OF ATTRIBUTE.&Type({AttrList}{@attrType}) }

Attributes { ATTRIBUTE:AttrList } ::= SET SIZE (1..MAX) OF Attribute {{ AttrList }}

END

```

## A.2 Module CMSObjectIdentifiers

```

CMSObjectIdentifiers
 {iso(1) identified-organization(3) tc68(133) country(16) x9(840)
 x9Standards(9) x9-73(73) module(0) oids(1) v2009(1)}

DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS All --

-- IMPORTS All --

OID ::= OBJECT IDENTIFIER -- Alias

-- Content types, from RSA PKCS #7 and IETF S/MIME

pkcs7 OID ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) }

id-data OID ::= { pkcs7 data(1) }

id-signedData OID ::= { pkcs7 signedData(2) }

id-envelopedData OID ::= { pkcs7 envelopedData (3) }

id-digestedData OID ::= { pkcs7 digestedData(5) }

id-encryptedData OID ::= { pkcs7 encryptedData (6) }

id-namedkeyencryptedData OID ::= { iso(1) member-body(2) us(840)
 x973(10060) types(1) namedKeyEncryptedData(2) }

-- Signcryption object identifiers --

id-signcryptData OID ::= itu-t recommendation(0) x(24) cms-profile(894) signcryption(1)
data(0)}

-- The signcryption-manifest arc is the root identifier for all --

-- SigncryptData manifest types defined in this standard. --

signcryption-manifest OID ::= { id-signcryptData manifest(1) }

-- A value of xPath identifies the XPathSet manifest type. --

xPath OID ::= { signcryption-manifest xPath(0) }

pkcs9 OID ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) }

smime OID ::= { pkcs9 smime(16) }

id-ct-authData OID ::= { smime ct(1) 2 }

```

```
-- Signed attributes, from RSA PKCS #9, IETF S/MIME, and X9.73 --
id-contentType OID ::= { pkcs9 contentType(3) }
id-messageDigest OID ::= { pkcs9 messageDigest(4) }

-- Authenticated attribute, from IETF S/MIME --

-- CKM key management object identifiers --
id-ckm-recipient-info OID ::= { iso member-body(2) us(840) x973(10060) km(2) 1 }
id-ckm-recipient-info2 OID ::= { iso member-body(2) us(840) x973(10060) km(2) 2 }
id-ckm-algorithms OID ::= { iso member-body(2) us(840) x973(10060) algorithms(3) }
id-ckm-symmetric OID ::= { id-ckm-algorithms symmetric(1) }
id-ckm-key-transport OID ::= { id-ckm-algorithms key-transport(2) }
id-ckm-key-agree-multiple-encrypt OID ::= {
id-ckm-algorithms key-agree-multiple-encrypt(3) }
id-ckm-key-agree-hash OID ::= { id-ckm-algorithms key-agree-hash(4) }
id-ckm-header OID ::= { iso member-body(2) us(840) x973(10060) header(4) }
 ckm-CMS OID ::= {
 joint-iso-itu-t(2) international-organizations(23) set(42) vendors(9)
 griffin(10) business(3) tecsec(0) cms(2) header(2) }

id-Ivec OID ::= { ckm-CMS 1 }
id-Secryptm OID ::= { ckm-CMS 2 }
id-Filelength OID ::= { ckm-CMS 3 }
id-Filehash OID ::= { ckm-CMS 4 }
id-Filename OID ::= { ckm-CMS 5 }
id-Domainlist OID ::= { ckm-CMS 6 }
id-Accessgrouplist OID ::= { ckm-CMS 7 }
id-Issuer OID ::= { ckm-CMS 8 }
id-Credentiallist OID ::= { ckm-CMS 9 }
id-SignKey OID ::= { ckm-CMS 10 }
id-KeyUsage OID ::= { ckm-CMS 11 }
id-BitSpray OID ::= { ckm-CMS 12 }
id-BitSprayMeta OID ::= { ckm-CMS 12 1 }
id-BitSprayShares OID ::= { ckm-CMS 12 2 }
id-FavoriteName OID ::= { ckm-CMS 13 }
id-DataSignature OID ::= { ckm-CMS 14 }
id-BlockSize OID ::= { ckm-CMS 15 }
id-DataFormat OID ::= { ckm-CMS 16 }

-- Tokenization object identifiers --
id-tokenization-manifest OID ::= { iso(1) identified-organization(3) tc68(133)
 country(16) x9(840) x9Standards(9) x9-73(73) tokenization(3) }
id-tokenizedParts OID ::= { id-tokenization-manifest tokenizedParts(0) }
id-XPathTokensSet OID ::= { id-tokenization-manifest xPathTokensSet(1) }

-- Signcrypt processing mode object identifiers --
signcrypt OID ::= { iso(1) identified-organization(3) tc68(133)
 country(16) x9(840) x9Standards(9) x9-73(73) signcrypt(4) }
signcrypt-mode OID ::= { signcrypt modes(1) }
```

```

signcrypt-content OID ::= { signcrypt-mode content(1) }
signcrypt-attributes OID ::= { signcrypt-mode attributes(2) }
signcrypt-components OID ::= { signcrypt-mode components(3) }
signcrypt-envelope OID ::= { signcrypt-mode enveloped(4) }
-- Signcrypt object identifiers --
id-signcryptParts OID ::= { signcrypt-manifest signcryptParts(1) }
id-XPathSigncryptSet OID ::= { signcrypt-manifest xpathSigncryptSet(2) }
-- X9.73 attribute object identifiers --
id-cms-attributes OID ::= { iso(1) identified-organization(3) tc68(133)
country(16) x9(840) x9Standards(9) x9-73(73) attributes(1) }
xmlMarkup OID ::= { id-cms-attributes xml(0) }
id-cms-SAML OID ::= { xmlMarkup saml(1) }
id-cms-XKMS OID ::= { xmlMarkup xkms(2) }
id-messageComponents OID ::= { xmlMarkup mc(3) }
-- X9.73 XML namespace prefix values --
id-cms-namespaces OID ::= { iso(1) identified-organization(3) tc68(133)
country(16) x9(840) x9Standards(9) x9-73(73) namespaces(2) }
cms OID ::= { iso(1) identified-organization(3) tc68(133) country(16) x9(840)
x9Standards(9) x9-73(73) namespaces(2) cms(0) }
-- RFC 3211 password-based encryption --
id-alg-PWRI-KEK OBJECT IDENTIFIER ::= { iso(1) member-body(2)
us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) alg(3) 9 }
-- Database Encryption Key Management --
dbEKM OID ::= { joint-iso-itu-t(2) country(16) us(840) organization(1) wfbna(114171)
lobs(4) eisArchitecture(1) techniques(2) dbEKM(0) }
id-SimpleString OID ::= { dbEKM ss(1) }
id-UniqueIdentifier OID ::= { dbEKM uid(2) }
id-dbekm-recv-info OID ::= { iso member-body(2) us(840) x973(10060) km(2) 3 }
END -- CMSObjectIdentifiers --

```

### A.3 Module AlgorithmInformation-2009 (from IETF RFC 5912)

```

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)}
DEFINITIONS EXPLICIT TAGS ::=
BEGIN
EXPORTS ALL;
IMPORTS

KeyUsage

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

-- Suggested prefixes for algorithm objects are:
--
-- mda- Message Digest Algorithms

```

```
-- sa- Signature Algorithms
-- kta- Key Transport Algorithms (Asymmetric)
-- kaa- Key Agreement Algorithms (Asymmetric)
-- kwa- Key Wrap Algorithms (Symmetric)
-- kda- Key Derivation Algorithms
-- maca- Message Authentication Code Algorithms
-- pk- Public Key
-- cea- Content (symmetric) Encryption Algorithms
-- cap- S/MIME Capabilities

ParamOptions ::= ENUMERATED {
 required, -- Parameters MUST be encoded in structure
 preferredPresent, -- Parameters SHOULD be encoded in structure
 preferredAbsent, -- Parameters SHOULD NOT be encoded in structure
 absent, -- Parameters MUST NOT be encoded in structure
 inheritable, -- Parameters are inherited if not present
 optional, -- Parameters MAY be encoded in the structure
 ...
}

-- DIGEST-ALGORITHM
--
-- Describes the basic information for ASN.1 and a digest
-- algorithm.
--
-- &id - contains the OID identifying the digest algorithm
-- &Params - if present, contains the type for the algorithm
-- parameters; if absent, implies no parameters
-- &paramPresence - parameter presence requirement
--
-- Additional information such as the length of the hash could have
-- been encoded. Without a clear understanding of what information
-- is needed by applications, such extraneous information was not
-- considered to be of sufficient importance.
--
-- Example:
-- mda-sha1 DIGEST-ALGORITHM ::= {
-- IDENTIFIER id-sha1
-- PARAMS TYPE NULL ARE preferredAbsent
-- }

DIGEST-ALGORITHM ::= CLASS {
 &id OBJECT IDENTIFIER UNIQUE,
 &Params OPTIONAL,
 &paramPresence ParamOptions DEFAULT absent
 } WITH SYNTAX {
 IDENTIFIER &id
 [PARAMS [TYPE &Params] ARE &paramPresence]
 }

-- SIGNATURE-ALGORITHM
--
-- Describes the basic properties of a signature algorithm
--
-- &id - contains the OID identifying the signature algorithm
-- &Value - contains a type definition for the value structure of
-- the signature; if absent, implies that no ASN.1
-- encoding is performed on the value
-- &Params - if present, contains the type for the algorithm
-- parameters; if absent, implies no parameters
-- &paramPresence - parameter presence requirement
-- &HashSet - The set of hash algorithms used with this
-- signature algorithm
-- &PublicKeySet - the set of public key algorithms for this
-- signature algorithm
-- &smimeCaps - contains the object describing how the S/MIME
-- capabilities are presented.
--
-- Example:
-- sig-RSA-PSS SIGNATURE-ALGORITHM ::= {
-- IDENTIFIER id-RSASSA-PSS
```