National Standard
for Financial Services

X9.96-2003

XML Cryptographic Message Syntax (XCMS)

Notice – This document is a draft document. It has not yet been processed through the consensus procedures of X9 and ANSI.

Many changes, which may greatly affect its contents, can occur before this document is completed. The X9F3 working group may not be held responsible for the contents of this document.

Implementation or design based on this revised draft standard is at the risk of the user. No advertisement or citation implying compliance with a “Standard” should appear, as it is erroneous and misleading to so state.

Copies of this revised draft proposed American National Standard will be available from the X9 Secretariat when the document is finally announced for two months public comment. Notice of this announcement will be in the trade press.

Secretariat:
Accredited Standards Committee X9, Incorporated

Approved:
American National Standards Institute

Foreword

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria for approval have been met by the standards developer.

Consensus is established when, in the judgment of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that a concerted effort be made toward their resolution.

The use of American National Standards is completely voluntary; their existence does not in any respect preclude anyone, whether he has approved the standards or not from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standards.

The American National Standards Institute does not develop standards and will in no circumstances give an interpretation of any American National Standard. Moreover, no person shall have the right or authority to issue an interpretation of an American National Standard in the name of the American National Standards Institute. Requests for interpretations should be addressed to the secretariat or sponsor whose name appears on the title page of this standard.

CAUTION NOTICE: This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken to reaffirm, revise, or withdraw this standard no later than five years from the date of approval.

Published by

Accredited Standards Committee X9, Incorporated
Financial Industry Standards
P. O. Box 4035
Annapolis, MD 21403
http://www.x9.org/

Contents

Introduction

NOTE: The user's attention is called to the possibility that compliance with this standard may require the use of an invention covered by patent rights.

By publication of this standard, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. The patent holder has, however, filed a statement of willingness to grant a license under these rights on reasonable and non-discriminatory terms and conditions to applicants desiring to obtain such a license. Details may be obtained from the standards developer.

Suggestions for the improvement or revision of this Standard are welcome. They should be sent to the X9 Committee Secretariat, American Bankers Association, 1120 Connecticut Avenue, N.W., Washington, D.C. 20036.

This Standard was processed and approved for submittal to ANSI by the Accredited Standards Committee on Financial Services, X9. Committee approval of the Standard does not necessarily imply that all the committee members voted for its approval.
Secretariat will provide current text for the following:

The X9 committee had the following members:

Gene Kathol, Chairman

Vincent DiSantis, Vice Chairman

Cynthia L. Fuller, Managing Director

Isabel Bailey, Program Manager

Organization Represented	**Representative**
ACI Worldwide	Cindy Rink
ACI Worldwide	Jim Shaffer
American Bankers Association	Don Rhodes
American Bankers Association	Stephen Schutze
American Bankers Association	Michael Scully
American Express Company	Mike Jones
American Express Company	Barbara Wakefield
American Financial Services Association	John Freeman
American Financial Services Association	Mark Zalewski
BancTec, Inc.	Rosemary Butterfield
BancTec, Inc.	Christopher Dowdell
BancTec, Inc.	David Hunt
Bank of America	Mack Hicks
Bank of America	Richard Phillips
Bank of America	Daniel Welch
Bank One Corporation	Jacqueline Pagan
BB and T	Michael Saviak
BB and T	Woody Tyner
Cable & Wireless America	Dr. William Hancock CISSP CISM
Cable & Wireless America	Shannon Myers
Cable & Wireless America	Kevin M. Nixon CISSP CISM

Cable & Wireless America	Jonathan Siegel
Citigroup, Inc.	Dan Schutzer
Citigroup, Inc.	Mark Scott
Citigroup, Inc.	Skip Zehnder
Deluxe Corporation	Maury Jansen
Diebold, Inc.	Bruce Chapa
Diebold, Inc.	Anne Doland
Diebold, Inc.	Judy Edwards
Discover Financial Services Inc.	Masood Mirza
eFunds Corporation	Chuck Bram
eFunds Corporation	Richard Fird
eFunds Corporation	Daniel Rick
eFunds Corporation	Joseph Stein
eFunds Corporation	Cory Surges
Electronic Data Systems	Linda Low
Federal Reserve Bank	Jeannine M. DeLano
Federal Reserve Bank	Dexter Holt
Federal Reserve Bank	Laura Walker
First Data Corporation	Gene Kathol
Fiserv	Bud Beattie
Fiserv	Kevin Finn
Fiserv	Dan Otten
Hewlett Packard	Larry Hines
Hewlett Packard	Gary Lefkowitz
IBM Corporation	Todd Arnold
Ingenico	John Sheets
Ingenico	John Spence
Inovant	Richard Sweeney
KPMG LLP	Tim Gartin
KPMG LLP	Mark Lundin
KPMG LLP	Jeff Stapleton
KPMG LLP	Al Van Ranst, Jr.
Mag-Tek, Inc.	Jeff Duncan
Mag-Tek, Inc.	Mimi Hart
Mag-Tek, Inc.	Carlos Morales
MasterCard International	Caroline Dionisio
MasterCard International	Naiyre Foster
MasterCard International	Ron Karlin
MasterCard International	William Poletti
Mellon Bank, N.A.	Richard Adams
Mellon Bank, N.A.	David Taddeo
National Association of Convenience Stores	John Hervey
National Association of Convenience Stores	Teri Richmond
National Association of Convenience Stores	Robert Swanson
National Security Agency	Sheila Brand
NCR Corporation	David Norris
NCR Corporation	Steve Stevens
Niteo Partners	Charles Friedman
Niteo Partners	Michael Versace
Silas Technologies	Andrew Garner
Silas Technologies	Ray Gatland
Star Systems, Inc.	Elizabeth Lynn
Star Systems, Inc.	Michael Wade
Symmetricom	John Bernardi

Symmetricom	Sandra Lambert
Symmetricom	Jerry Willett
The Clearing House	Vincent DeSantis
The Clearing House	John Dunn
Unisys Corporation	David J. Concannon
Unisys Corporation	Navnit Shah
VeriFone	David Ezell
VeriFone	Dave Faoro
VeriFone	Allison Holland
VeriFone	Brad McGuinness
VeriFone	Brenda Watlington
VISA International	Patricia Greenhalgh
Wells Fargo Bank	Terry Leahy
Wells Fargo Bank	Gordon Martin

The X9F subcommittee on Data and Information Security had the following members:

Richard Sweeney, Chair, Inovant

Organization	**Representative**
3PEA Technologies, Inc.	Mark Newcomer
3PEA Technologies, Inc.	Daniel Spence
ACI Worldwide	Cindy Rink
ACI Worldwide	Jim Shaffer
American Bankers Association	Don Rhodes
American Bankers Association	Stephen Schutze
American Express Company	William J. Gray
American Express Company	Mike Jones
American Express Company	Mark Merkow
American Financial Services Association	John Freeman
American Financial Services Association	Mark Zalewski
BancTec, Inc.	Rosemary Butterfield
Bank of America	Mack Hicks
Bank of America	Todd Inskeep
Bank of America	Richard Phillips
Bank of America	Daniel Welch
Bank of America	Craig Worstell
Bank One Corporation	Jacqueline Pagan
BB and T	Michael Saviak
BB and T	Woody Tyner
Cable & Wireless America	Dr. William Hancock CISSP CISM
Cable & Wireless America	Shannon Myers
Cable & Wireless America	Kevin M. Nixon CISSP CISM
Cable & Wireless America	Jonathan Siegel
Certicom Corporation	Daniel Brown
Communications Security Establishment	Mike Chawrun
Communications Security Establishment	Alan Poplove
Deluxe Corporation	Maury Jansen
Diebold, Inc.	Bruce Chapa
Diebold, Inc.	Anne Doland
Diebold, Inc.	Judy Edwards
Discover Financial Services Inc.	Pamela Ellington
Discover Financial Services Inc.	Masood Mirza

Diversinet Corporation	Michael Crerar
eFunds Corporation	Chuck Bram
Electronic Industries Alliance	Edward Mikoski
Electronic Industries Alliance	Donald L. Skillman
Entrust Technologies	Miles Smid
Federal Reserve Bank	Neil Hersch
Ferris and Associates, Inc.	J. Martin Ferris
First Data Corporation	Gene Kathol
Fiserv	Bud Beattie
Fiserv	Dan Otten
Hewlett Packard	Larry Hines
Hewlett Packard	Gary Lefkowitz
IBM Corporation	Todd Arnold
IBM Corporation	Michael Kelly
IBM Corporation	Allen Roginsky
Identrus	Brandon Brown
Ingenico	John Sheets
Ingenico	John Spence
Inovant	Richard Sweeney
International Biometric Group	Mcken Mak CISSP
International Biometric Group	Michael Thieme
Jones Futurex, Inc.	Ray Bryan
Jones Futurex, Inc.	Scott Davis
Jones Futurex, Inc.	Barry Golden
Jones Futurex, Inc.	Steve Junod
KPMG LLP	Tim Gartin
KPMG LLP	Mark Lundin
KPMG LLP	Jeff Stapleton
KPMG LLP	Al Van Ranst, Jr.
Mag-Tek, Inc.	Terry Benson
Mag-Tek, Inc.	Mimi Hart
MasterCard International	Ron Karlin
MasterCard International	William Poletti
Mellon Bank, N.A.	David Taddeo
National Association of Convenience Stores	John Hervey
National Association of Convenience Stores	Teri Richmond
National Association of Convenience Stores	Robert Swanson
National Security Agency	Sheila Brand
NCR Corporation	Wayne Doran
NCR Corporation	Charlie Harrow
NCR Corporation	David Norris
NCR Corporation	Steve Stevens
Niteo Partners	Charles Friedman
Niteo Partners	Michael Versace
NIST	Elaine Barker
NIST	Lawrence Bassham III
NIST	Morris Dworkin
NIST	Annabelle Lee
NTRU Cryptosystems, Inc.	Ari Singer
NTRU Cryptosystems, Inc.	William Whyte
Pitney Bowes, Inc.	Matthew Campagna
Pitney Bowes, Inc.	Andrei Obrea
Pitney Bowes, Inc.	Leon Pintsov
R Squared Academy Ltd.	Richard E. Overfield Jr.

R Squared Academy Ltd.	Ralph Spencer Poore
RSA Securities	Burt Kaliski
Star Systems, Inc.	Elizabeth Lynn
Star Systems, Inc.	Michael Wade
Surety, Inc.	Dimitrios Andivahis
Symmetricom	Sandra Lambert
TECSEC Incorporated	Pud Reaver Y3YD
TECSEC Incorporated	Ed Scheidt
TECSEC Incorporated	Dr. Wai L. Tsang, Ph.D.
TECSEC Incorporated	Jay Wack
Thales e-Security, Inc.	Paul Meadowcroft
Thales e-Security, Inc.	Brian Sullivan
Thales e-Security, Inc.	James Torjussen
VeriFone.	Dave Faoro
VeriFone	Brad McGuinness
VISA International	Patricia Greenhalgh
VISA International	Richard Hite
Wells Fargo Bank	Terry Leahy
Wells Fargo Bank	Gordon Martin
Wells Fargo Bank	Ruven Schwartz

Under ASC X9 procedures, a working group may be established to address specific segments of work under the ASC X9 Committee or one of its subcommittees. A working group exists only to develop standard(s) or guideline(s) in a specific area and is then disbanded. The individual experts are listed with their affiliated organizations. However, this does not imply that the organization has approved the content of the standard or guideline. (Note: Per X9 policy, company names of non-member participants are listed only if, at time of publication, the X9 Secretariat received an original signed release permitting such company names to appear in print.)

The X9F3 Working Group, which developed this standard had the following members:

C. L. Reaver, Chair

Phillip H. Griffin, Technical Editor

pecial thanks to Phil Griffin for his contributions to the text, ASN.1 and XML

Organization	**Representative**
3PEA Technologies, Inc.	Mark Newcomer
3PEA Technologies, Inc.	Daniel Spence
American Express Company	Mike Jones
Bank of America	Andi Coleman
Bank of America	Todd Inskeep
Cable & Wireless America	Dr. William Hancock CISSP CISM
Cable & Wireless America	Shannon Myers
Cable & Wireless America	Kevin M. Nixon CISSP CISM
Cable & Wireless America	Jonathan Siegel
Certicom Corporation	Daniel Brown
Certicom Corporation	John O. Goyo
Diebold, Inc.	Bruce Chapa
Diebold, Inc.	Anne Doland
Diebold, Inc.	Judy Edwards
eFunds Corporation	Chuck Bram

Entrust Technologies	Don Johnson
Entrust Technologies	Miles Smid
Entrust Technologies	Robert Zuccherato
Ernst and Young	Keith Sollers
Federal Reserve Bank	Neil Hersch
First Data Corporation	Curt Beeson
First Data Corporation	Lisa Curry
First Data Corporation	Lynn Wheeler
Fiserv	Dan Otten
Gilbarco	Tim Weston
Hewlett Packard	Larry Hines
IBM Corporation	Todd Arnold
IBM Corporation	Michael Kelly
Ingenico	John Sheets
Ingenico	John Spence
Inovant	Richard Sweeney
Jones Futurex, Inc.	Jason Anderson
Jones Futurex, Inc.	Ray Bryan
Jones Futurex, Inc.	Steve Junod
KPMG LLP	Tim Gartin
KPMG LLP	Jeff Stapleton
Mag-Tek, Inc.	Terry Benson
MasterCard International	William Poletti
National Security Agency	Sheila Brand
National Security Agency	Greg Gilbert
National Security Agency	Tim Havighurst
National Security Agency	Paul Timmel
NCR Corporation	Steve Stevens
Niteo Partners	Charles Friedman
Niteo Partners	Michael Versace
NIST	Elaine Barker
NTRU Cryptosystems, Inc.	Ari Singer
NTRU Cryptosystems, Inc.	William Whyte
PNC Bank, NA	Tim Garland
Pulse EFT Association.	Vivian M. Banki
Pulse EFT Association.	Donald Rickett
R Squared Academy Ltd.	Richard E. Overfield Jr.
R Squared Academy Ltd.	Ralph Spencer Poore
Surety, Inc.	Dimitrios Andivahis
Symmetricom	Sandra Lambert
TECSEC Incorporated	Pud Reaver Y3YD
TECSEC Incorporated	Ed Scheidt
TECSEC Incorporated	Dr. Wai L. Tsang
Thales e-Security, Inc.	Tim Fox
Thales e-Security, Inc.	Brian Sullivan
Thales e-Security, Inc.	James Torjussen
VeriFone.	Dave Faoro
VISA International	Richard Hite

XML Cryptographic Message Syntax (XCMS)

1 Scope

This Standard specifies a text based Cryptographic Message Syntax (CMS) represented using XML 1.0 encoding that can be used to protect financial transactions and other documents from unauthorized disclosure and modification. The message syntax has the following characteristics:

1) Protected messages are represented using the Canonical XML Encoding Rules (cXER), and can be transferred as verbose markup text or in a compact, efficient binary representation using the Basic Encoding Rules (BER) or the canonical subset of BER, the Distinguished Encoding Rules (DER).

2) Messages are protected independently. There is no cryptographic sequencing (e.g., cipher block chaining) between messages. There need not be any real-time connection between the sender and recipient of the message. This makes the syntax suitable for use over store-and-forward systems, e.g. Automated Clearing House (ACH) or Society for Worldwide International Funds Transfer (SWIFT). Standard attributes are defined to allow applications to maintain relationships between messages, if desired.

3) The syntax is algorithm independent. It supports confidentiality, integrity, origin authentication, and non-repudiation services. Only ANSI X9-approved algorithm(s) may be used for message digest, message encryption, digital signature, message authentication, and key management.

4) Support for biometric security, enhanced certificate techniques such as compact domain certificates and key management extensions such as Constructive Key Management (CKM) are provided.

5) Selective field protection can be provided in two ways. First by combining multiple instances of this syntax into a composite message. And second by using identifier and type markup tag names to select message components to be protected in a single message, which allows reusable message components to be moved between documents without affecting the validity of the signature.

6) Precise message encoding and cryptographic processing requirements are provided.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. Nevertheless, parties to agreements based on this document are encouraged to consider applying the most recent editions of the referenced documents indicated below. For undated references, the latest edition of the referenced document (including any amendments and technical corrections) applies.

[1] ANS X9.19-1996 *Financial Institution Retail Message Authentication (MAC)*

[2] ANS X9.30-1997 *Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry, Part 1: The Digital Signature Algorithm (DSA)*

[3] ANS X9.30-1997 *Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry, Part 2: The Secure Hash Algorithm (SHA)*[4] ANS X9.31-1998 *Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry: The RSA Signature Algorithm*

[4] ANS X9.31-1998 *Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry: The RSA Signature Algorithm*

[5] ANS X9.42-2001, Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography.

[6] ANS X9.45-1997, Enhanced Management Controls Using Digital Signatures and Attribute Certificates.

[7] ANS X9.62-1999 Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)

[8] ANS X9.63-2001, Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.

[9] ANS X9.68-2001, Digital Certificates for Mobile/Wireless and High Transaction Volume Financial Systems: Part 2: Domain Certificate Syntax.

[10] ANS X9.69-1999, Framework for Key Management Extensions.

[11] ANS X9.71-1999 Keyed Hash Message Authentication Code (HMAC)

[12] ANS X9.73-2002, Cryptographic Message Syntax (CMS).

[13] ANS X9.84-2003, Biometric Information Management and Security.

[14] ISO/IEC ISO/IEC 8824-1 | ITU-T Recommendation X.680, Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation.

[15] ISO/IEC 8824-2 | ITU-T Recommendation X.681, Information Technology - Abstract Syntax Notation One (ASN.1): Information Object Specification.

[16] ISO/IEC 8824-3 | ITU-T Recommendation X.682, Information Technology - Abstract Syntax Notation One (ASN.1): Constraint Specification.

[17] ISO/IEC 8824-4 | ITU-T Recommendation X.683, Information Technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 Specifications.

[18] ISO/IEC 8825-1 | ITU-T Recommendation X.690, Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

[19] ISO/IEC 8825-4 | ITU-T Recommendation X.693, Information Technology - ASN.1 Encoding Rules: Specification of XML Encoding Rules (XER).

[20] W3C XML 1.0:2000, *Extensible Markup Language (XML) 1.0 (Second Edition),* W3C Recommendation, Copyright © [6 October 2000] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), *http://www.w3.org/TR/2000/REC-xml-20001006.*

3 Terms, definitions, symbols and abbreviated terms

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

3.1
Abstract Syntax Notation One
ASN.1
A notation that is used in describing messages to be exchanged between communicating application programs. ASN.1 is used in this standard to describe the XML Cryptographic Message Syntax schema and transfer syntax using the ASN.1 Distinguished Encoding Rules (DER) [12] and XML Encoding Rules (XER) [13].

3.2
asymmetric cryptographic algorithm
A cryptographic algorithm that has two related keys, a public key and a private key; the two keys have the property that, given the public key, it is computationally infeasible to derive the private key.

3.3
certificate
digital certificate
The public key and identity of an entity, together with some other information, that is rendered unforgeable by signing the certificate with the private key of the Certification Authority that issued the certificate.

3.4
Certificate Authority
CA
An entity trusted by one or more other entities to create and assign certificates.

3.5
certificate revocation list
CRL
A list of digital certificates that have been revoked for one reason or another – usually because of compromise.

3.6
constructive key management
CKM
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 together using a mathematical function to produce an object key.

3.7
content encryption key
CEK
The symmetric key used to encrypt the content of a message.

3.8
cryptographic hash function
hash
A (mathematical) function that maps values from a large (possibly very large) domain into a smaller range. The function satisfies the following properties:

1. (One-way) It is computationally infeasible to find any input that maps to any pre-specified output;

2. (Collision Free) It is computationally infeasible to find any two distinct inputs that map to the same output.

3.9
cryptographic key
key
A parameter that determines, possibly with other parameters, the operation of a cryptographic function such as:

(a) the transformation from plaintext to ciphertext and vice versa;

(b) the synchronized generation of keying material;

(c) digital signature computation or validation.

3.10
cryptography
The discipline that embodies principles, means and methods for the transformation of data to hide its information content, prevent its undetected modification, prevent its unauthorized use or a combination thereof.

3.11
domain parameters
The prime p that defines GF(p), a prime factor q of $p-1$, and an associated generator g of order q in the multiplicative group $GF(p)^*$. These parameters are used to facilitate the use of algorithms based on discrete logarithm cryptography.

3.12
ephemeral key
A private or public key that is unique for each execution of a cryptographic scheme. An ephemeral private key is to be destroyed as soon as computational need for it is complete. An ephemeral public key may or may not be certified. In this standard, an ephemeral public key is represented by t, while an ephemeral private key is represented by r, with a subscript to represent the owner of the key.

3.13
forward secrecy
perfect forward secrecy
The assurance provided to an entity that the session key established with another entity will not be compromised by the compromise of either entity's static private key in the future.

3.14
key agreement
A method of establishing a key, whereby both parties contribute to the value of the resulting key and neither party can control the value of the resulting key.

3.15
key encryption key
A key used exclusively to encrypt and decrypt keys.

3.16
keying material
The data (e.g., keys, certificates and initialization vectors) necessary to establish and maintain cryptographic keying relationships.

3.17
key management
The generation, storage, secure distribution and application of keying material in accordance with a security policy.

3.18
key pair
When used in public key cryptography, a public key and its corresponding private key.

3.19
key transport
A key establishment protocol under which the secret key is determined by the initiating party.

3.20
message authentication code
MAC
A cryptographic value that is the result of passing a message through the message authentication algorithm using a specific key.

3.21
Multipurpose Internet Mail Extensions
MIME
The format for internet message bodies as defined in the IETF documents RFC 2045, RFC 2046, RFC 2047, RFC 2048 and RFC 2049.

3.22
nonce
A nonrepeating value, such as a counter, using key management protocols to thwart replay and other types of attack.

3.23
object
That which is to be encrypted.[1]

3.24
object key
A key used to encrypt and decrypt an object.

3.25
private key
In an asymmetric (public) key cryptosystem, the key of an entity's key pair that is known only by that entity. A private key may be used:

(1) to compute the corresponding public key;

(2) to make a digital signature that may be verified by the corresponding public key;

(3) to decrypt data encrypted by the corresponding public key; or

(4) together with other information to compute a piece of common shared secret information.

3.26
public key
In an asymmetric (public) key cryptosystem, that key of an entity's key pair that may be publicly known. A public key may be used:

[1] When using CKM.

(1) to verify a digital signature that is signed by the corresponding private key;

(2) to encrypt data that may be decrypted by the corresponding private key;

(3) by other parties to compute a piece of shared information.

3.28
Secure MIME
S/MIME
The specification for handling MIME data securely by adding cryptographic security services to supply authentication, message integrity, non-repudiation of origin, privacy and data security. The specification is found in IETF documents RFC 2311 and 2312. See Multipurpose Internet Mail Extensions (MIME).

3.29
shared symmetric key
A symmetric key derived from a shared secret value and other information.

3.30
static key
A private or public key that is common to many executions of a cryptographic scheme. A static public key may be certified. In this standard, the letter "*y*" represents a static public key, while a static private key is represented by "*x*", each with a subscript to represent the owner of the key. See definition of ephemeral key.

3.31
symmetric cryptographic algorithm
A cryptographic algorithm that uses one shared key, a secret key. The key must be kept secret between the two communicating parties. The same key is used for both encryption and decryption.

3.32
symmetric key
A cryptographic key that is used in symmetric cryptographic algorithms. The same symmetric key that is used for encryption is also used for decryption.

4 Organization

The following normative annexes are integral parts of the standard that, for reasons of convenience, are placed after all normative elements.

Annex	Contents	Normative/Informative
A	ASN.1 Module for Object Identifiers	Normative
B	X9.96 XCMS Schema	Normative

5 Application

The XML cryptographic message syntax defined in this standard provides the same security services provided in the ANS X9.73 [12] standard. This includes the following:

1) Independent data unit protection, where each message or transaction is protected independently. There is no need for a real-time communications session between the sender and recipient, and no cryptographic sequencing (such as cipher block chaining) between messages. This standard does define attributes that allow applications to maintain relationships between messages;

2) Confidentiality, using any ANSI X9 approved symmetric encryption algorithm and any ANSI X9 approved key management algorithm. Typically, the key management algorithm is used to protect a content-encryption key used to encrypt the message. This approach allows the sender to send an encrypted message to multiple recipients, while only encrypting the actual message once. The syntax is optimized for the common case where the same key management algorithm and parameters are used for all recipients;

3) Data integrity origin authentication of data, using any ANSI X9 approved digital signature or message authentication algorithm. (When using digital signatures, non-repudiation may also be supported.) Support for multiple signers, per-signer authenticated attributes, unsigned attributes, and countersignatures, are also provided. An optimized syntax is also provided for the common case where only a single entity signs or authenticates a message.

Unlike ANS X9.73, this syntax allows for both the selective protection of specific fields within a message, or protection of the entire message. Message protection of selected specific fields can also be implemented by combining multiple protected messages into a composite message. In general, selective field protection requires knowledge of the message, and this information must be included in a signed attribute.

This syntax specifies an XML [20] encoding of the enhanced cryptographic message syntax defined in ANS X9.73. Additional attributes for use in financial applications, as well as cryptographic processing required for use with ANSI X9 approved cryptographic algorithms and on XML markup plaintext messages are defined.

6 Message Structures

6.1 Encapsulated Content

The message structures in this standard are defined for transfer using XML markup or compact binary encodings. The cryptographic message schema is defined using Abstract Syntax Notation One [14], [15], [16], [17], and the XML markup specified in this standard conforms to the XML Encoding Rules (XER) [19] of ASN.1. The following subsections describe the XML Cryptographic Message Syntax (XCMS) protected message types. A full specification of the schema for the XML markup in this standard can be found in Annex B. This schema can also be used to generate compact binary encodings using the Distinguished Encoding Rules (DER) [18] of ASN.1.

XCMS associates a content type identifier with a content type. The associated content type is wrapped in a value of type **`OCTET STRING`**, an “octet hole”, which contains the complete encoding of a value of an ASN.1 type. The content identifier and content type form a value of type **`EncapsulatedContentInfo`** defined as:

```
EncapsulatedContentInfo ::= SEQUENCE {
   eContentType  ContentType,
   eContent      [0] EXPLICIT
                    CONTENTS.&Type({Contents}{@eContentType})  OPTIONAL
}
```

Type **`EncapsulatedContentInfo`** is composed of two components, **`eContentType`** and **`eContent`**. The **`eContentType`** value is an object identifier, which indicates the type of content encapsulated in the **`eContent`** component. The **eContent** component is an octet hole containing content identified by the **eContentType** component.

Type **ContentType** is defined in terms of the **&id** field of the **CONTENTS** information object set:

```
ContentType ::= CONTENTS.&id({Contents})

CONTENTS ::= TYPE-IDENTIFIER  -- Defined in ISO/IEC 8824-2, Annex A
```

Content types for data, signed-data, enveloped-data, authenticated-data, digested-data, encrypted-data and named-key-encrypted-data are defined in this standard. A value of **ContentType** is a unique object identifier from the information object set **Contents**. The **Contents** information object set imposes a constraint on the valid values of **ContentType**. This set of objects is defined as:

```
Contents CONTENTS ::= {
   { ESignedData             IDENTIFIED BY  id-signedData            } |
   { EEnvelopedData          IDENTIFIED BY  id-envelopedData         } |
   { EAuthenticatedData      IDENTIFIED BY  id-ct-authData           } |
   { EDigestedData           IDENTIFIED BY  id-digestedData          } |
   { EEncryptedData          IDENTIFIED BY  id-encryptedData         } |
   { ENamedKeyEncryptedData  IDENTIFIED BY  id-namedkeyencryptedData } |
   { EData                   IDENTIFIED BY  id-data                  },

   ...  -- Expect additional objects --
}
```

Each object identifier in the **Contents** set is paired with an octet string that contains an ASN.1 type defined as:

```
ESignedData ::= OCTET STRING (CONTAINING SignedData)

EEnvelopedData ::= OCTET STRING (CONTAINING EnvelopedData)

EAuthenticatedData ::= OCTET STRING (CONTAINING AuthenticatedData)

EDigestedData ::= OCTET STRING (CONTAINING DigestedData)

EEncryptedData ::= OCTET STRING (CONTAINING EncryptedData)

ENamedKeyEncryptedData ::= OCTET STRING (CONTAINING NamedKeyEncryptedData)

EData ::= OCTET STRING (CONTAINING Data)
```

The following table illustrates the relation between the set of valid object identifiers and the encapsulated ASN.1 types that they identify:

Object Identifier Name	Identified Type Name	Encapsulated Type Name
id-signedData	SignedData	EsignedData
id-envelopedData	EnvelopedData	EenvelopedData
id-ct-authData	AuthenticatedData	EauthenticatedData
id-digestedData	DigestedData	EdigestedData

id-encryptedData	EncryptedData	EencryptedData
id-namedkeyencryptedData	NamedKeyEncryptedData	EnamedKeyEncryptedData
id-data	Data	Edata

A value of type **EncapsulatedContentInfo** can be represented using XML markup as

```
<enCapContentInfo>
   <eContentType> 1.2.840.113549.1.7.2 </eContentType>
   <eContent>
      <SignedData>
          ...
      </SignedData>
   </eContent>
</enCapContentInfo>
```

Here the **eContentType** indicates that the complete encoding of a value of type **SignedData** is encapsulated in the **eContent** component. In this example, an ellipsis is used as a placeholder for the **SignedData** components.

The object identifier values that identify the content types in this standard are defined below. All of the object identifiers defined in this standard are based on an alias for values of type **OBJECT IDENTIFIER**, the defined type **OID**[2].

```
id-data ::= <OID> 1.2.840.113549.1.7.1 </OID>

id-signedData ::= <OID> 1.2.840.113549.1.7.2 </OID>

id-envelopedData ::= <OID> 1.2.840.113549.1.7.3 </OID>

id-digestedData ::= <OID> 1.2.840.113549.1.7.5 </OID>

id-encryptedData ::= <OID> 1.2.840.113549.1.7.6 </OID>

id-ct-authData ::= <OID> 1.2.840.113549.1.9.16.1.2 </OID>

id-namedkeyencryptedData ::= <OID> 1.2.840.10060.1.2 </OID>
```

The id-data content type identifies opaque information, such as ASCII text, word processing files, spreadsheets, biometric information, or any other type of data whose structural details and interpretation are left to the application. Applications may use context to determine the actual type of underlying data, or MIME processing may be required to determine the actual content type.

The message types defined in this standard may be nested recursively to provide multiple security services. For example, to provide confidentiality, authentication, and integrity, the sender would typically create an instance of encrypted-data, and use it as the content for an instance of signed-data.

---

[2] The ASN.1 XML Value Notation is used in this standard where possible to demonstrate how values are encoded as XML markup. The more verbose and familiar Basic Value Notation is used to specify the same values in the complete ASN.1 modules found in the normative annexes.

The following sections describe signed-data, enveloped-data, authenticated-data, digested-data and encrypted-data.

6.2 Signed Data

6.2.1 Schema Definition

The **`SignedData`** type may consist of message content and one or more signatures and sets of certificates and Certificate Revocation Lists (CRLs) that can be used in signature verification. The **`SignedData`** type may also have no message content and no signers, and used in this manner as a mechanism to distribute certificates and CRLs.

For each signer, a **`SignedData`** value may include the following:

- a set of certificates and a set of CRLs needed to verify the signer's signature, and validate the signer's certificate.
- a set of attributes protected with the message content by the signer's signature;
- the signer's signature;
- a set of unsigned attributes.

The signed-data message provides origin authentication, data integrity, and (with appropriate additional measures such as auditing and accurate time-stamping), non-repudiation.

The message recipient uses a certificate identifier in a signature block to locate the certificate(s) needed to verify the signature.

The **`SignedData`** type is defined as:

```
SignedData ::= SEQUENCE {
   version           Version,
   digestAlgorithms  DigestAlgorithmIdentifiers,
   encapContentInfo  EncapsulatedContentInfo,
   certificates      [0] CertificateSet  OPTIONAL,
   crls              [1] CertificateRevocationLists  OPTIONAL,
   signerInfos       SignerInfos
}
```

The **`version`** component is an integer value that identifies the schema version number. Type **`Version`** is defined as:

```
Version ::= INTEGER { vx9-96(96) } ( vx9-96, ... )
```

The **`version`** for the schema in this standard is **`vx9-96`**. The extension marker, "…" allows any other version identifier to be used in an application.

The **`digestAlgorithms`** component is a value of type **`DigestAlgorithmIdentifiers`**, a collection of zero or more message digest algorithm identifiers. Each element in the collection identifies the message digest algorithm and any associated parameters used by one or more signer. Only ANSI X9 approved cryptographic hash algorithms are supported. Type **`DigestAlgorithmIdentifiers`** is defined as:

```
DigestAlgorithmIdentifiers ::= SET SIZE(0..MAX) OF DigestAlgorithmIdentifier

DigestAlgorithmIdentifier ::= AlgorithmIdentifier {{ DigestAlgorithms }}

DigestAlgorithms ALGORITHM ::= {
   SHA-Algorithms,

   ... -- Expect other digest algorithms  --
}
```

The **encapContentInfo** component is a value of type **EncapsulatedContentInfo**, which identifies and optionally carries the signed content. The **eContentType** component of type **EncapsulatedContentInfo** is an object identifier value that indicates the content type. When present, the optional **eContent** component contains the message content. This component may be absent to allow construction of “detached signatures”, but when the **eContent** value is absent, the signer calculates the signature on the message content as though the value were present.

The **certificates** component is a value of type **CertificateSet**, a collection of one or more certificates. This type is treated as an opaque string in this standard, and defined as:

```
CertificateSet ::= OCTET STRING
```

The certificates used in this standard are signed binary objects, whose digital signatures have been calculated over values encoded using the Distinguished Encoding Rules (DER) of ASN.1 using the schema defined for these types in ANS X9.73. In order to verify the signatures on these objects, their original encodings must be maintained. But these values must also be represented in XML encodings in a useful textual format. So the values in the **certificates** component of type **CertificateSet** have been Base64 armored to minimize their size when represented using XML markup while preserving their original encodings. The input to the Base64 processing is defined in ANS X9.73 as a Basic Encoding Rules (BER) encoded value of type **SET OF CertificateChoices**.

Any combination of ANS X9.68 [9] domain certificates, X.509 [24] certificates and attribute certificates may be included in the **CertificateSet** type, and they may appear in any order. There may be more or fewer certificates than needed for any purpose. Certificates are provided as needed to support key management techniques used in this standard. Use of the **CertificateSet** type to distribute certificates is not required. They may be obtained by other means, or an online certificate validation service may be used instead. Only version one ANS X9.68 domain certificates, version three X.509 certificates and version two attribute certificates are supported in this standard, to meet the needs of the financial services community as described in [22] and [23].

The **crls** component is a value of type **CertificateRevocationLists**, a collection of one or more CRLs. This type is treated as an opaque string in this standard, and defined as:

```
CertificateRevocationLists ::= OCTET STRING
```

The CRLs used in this standard are signed binary objects, whose digital signatures have been calculated over values encoded using the Distinguished Encoding Rules (DER) of ASN.1 using the schema defined for these types in ANS X9.73. In order to verify the signatures on these objects, their original encodings must be maintained. But these values must also be represented in XML encodings in a useful textual format. So the values in the **crls** component of type **CertificateRevocationLists** have been Base64 armored to minimize their size when represented using XML markup while preserving their original encodings. The input to the Base64 processing is defined in ANS X9.73 as a Basic Encoding Rules (BER) encoded value of type **SET OF CertificateList**. Any number of CRLs may be included in the **CertificateRevocationLists** type, and they may appear in any order. There may be more or fewer CRLs than needed for any purpose. CRLs are provided as needed to support certificate validation. Use of the **CertificateRevocationLists** type to distribute CRLs is not required. CRLs may be obtained by other means, or an online certificate validation service may be used instead. Only version two certificate revocation lists are supported in this standard.

Signer Information

Information about individual signers is represented in type **`SignerInfo`**.

```
SignerInfos ::= SET OF SignerInfo

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

The value of **`version`** is the schema version number. This value shall be ninety-six for this standard.

The **`sid`** component of **`SignerInfo`** identifies the signer's certificate. This standard provides three alternatives for identifying the signer's public key – **`issuerAndSerialNumber`**, **`subjectKeyIdentifier`**, and **`certHash`**.

```
SignerIdentifier ::= CHOICE {
   issuerAndSerialNumber  IssuerAndSerialNumber,
   subjectKeyIdentifier   [0] SubjectKeyIdentifier,
   certHash               [73] EXPLICIT Hash
}
```

The **`issuerAndSerialNumber`** choice alternative of type **`SignerIdentifier`** identifies the signer's X.509 certificate by the certificate's issuer distinguished name and serial number. ANS X9.68 domain certificates are uniquely identified by their owner names. They do not have an issuer-distinguished name and serial number, so this choice alternative may not be used to identify ANS X9.68 certificates. Type **`IssuerAndSerialNumber`** is defined as:

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

The **`subjectKeyIdentifier`** choice alternative of type **`SignerIdentifier`** identifies the signer's certificate by the X.509 or ANS X9.68 domain certificate **`subjectKeyIdentifier`** extension value.

```
SubjectKeyIdentifier ::= OCTET STRING
```

The **`certHash`** choice alternative of type **`SignerIdentifier`** can be used to identify any certificate format using the hash of the entire certificate. This alternative is a value of type **`Hash`** defined as:

```
Hash ::= CHOICE {
   ietf       CertHash,  -- SHA-1 hash of entire certificate
   withAlgID  DigestInfo
}
```

```
CertHash ::= OCTET STRING (ENCODED BY sha-1)

DigestInfo ::= SEQUENCE {
   hashAlgorithm  DigestAlgorithmIdentifier,
   digest         OCTET STRING
}
```

Type **Hash** offers two choice alternatives. The **ietf** alternative requires a SHA-1 [3] hash, and the **withAlGID** alternative allows any X9 approved hash algorithm to be used.

The **digestAlgorithm** component of type **SignerInfo** identifies the X9 approved message digest algorithm used by the content signer, and any associated algorithm parameters.

The **signedAttrs** component of type **SignerInfo** is a collection of attributes that are signed along with the message content. This component shall be present if the content type of the **EncapsulatedContentInfo** value being signed is not ordinary data identified by **id-data.**

The **signatureAlgorithm** component of type **SignerInfo** identifies the X9 approved signature algorithm, DSA [2], RSA [4], or ECDSA [7] and any associated parameters used by the content signer to generate the digital signature.

```
SignatureAlgorithmIdentifier ::= AlgorithmIdentifier {{ SignatureAlgorithms }}

SignatureAlgorithms ALGORITHM ::= {
   { OID dsa-with-sha1          PARMS NullParms } |
   { OID ecdsa-with-SHA1        PARMS NullParms } |
   { OID sha1WithRSAEncryption  PARMS NullParms },

   ...  -- Expect other signature algorithms --
}
```

The **signature** component of type **SignerInfo** is the digital signature on the **eContent** value (and any signed attributes) using the signer's **digestAlgorithm** and private key.

```
SignatureValue ::= OCTET STRING
```

The **unsignedAttrs** component of type **SignerInfo** is a collection of attributes that are not signed.

A value of type **SignedData** can be encoded using XML markup as:

```
<SignedData>
   <version> 96 </version>
   <digestAlgorithms>
      <DigestAlgorithmIdentifier>
         <algorithm> 1.3.14.3.2.26 </algorithm>
         <parameters> <NullParms/> </parameters>
      </DigestAlgorithmIdentifier>
   </digestAlgorithms>
   <encapContentInfo>
      <eContentType> 1.2.840.113549.1.7.1 </eContentType>
   </encapContentInfo>
   <signerInfos>
      <SignerInfo>
         <version> 96 </version>
```

```
        <sid>
           <certHash>
              <withAlgID>
                 <hashAlgorithm>
                    <algorithm> 1.3.14.3.2.26 </algorithm>
                    <parameters> <NullParms/> </parameters>
                 </hashAlgorithm>
                 <digest>
                   E6E66A9245BCD6749F43C1A16D270BAF249B70CA
                 </digest>
              </withAlgID>
           </certHash>
        </sid>
        <digestAlgorithm>
           <algorithm> 1.3.14.3.2.26 </algorithm>
           <parameters> <NullParms/> </parameters>
        </digestAlgorithm>
        <signatureAlgorithm>
           <algorithm> 1.2.840.10040.4.3 </algorithm>
           <parameters> <NullParms/> </parameters>
        </signatureAlgorithm>
        <signature>
           302C02144F9CA4507E2638AE9B632A3698A7AE84858F13
           3802140BD484312B36B090D2DF8B8A4719353F9A1EFAA5
        </signature>
     </SignerInfo>
  </signerInfos>
</SignedData>
```

Here the value in the `<version>` elements identifies this value of type **SignedData** as conforming to this standard. The `<digestAlgorithms>` element shows a single digest algorithm is used, a SHA-1 hash. TBS

6.2.2 Signed Attributes

This section defines a number of useful signed attributes. Applications are free to define their own attributes as well (see ANS X9.45 [6] for examples).

```
SignedAttributes ::= SET SIZE(1..MAX) OF SignedAttribute

SignedAttribute ::= Attribute {{Signed}}

Attribute { ATTRIBUTE:IOSet } ::= SEQUENCE {
   type    ATTRIBUTE.&id({IOSet}),
   values  SET OF ATTRIBUTE.&Type({IOSet}{@type})
}

Signed ATTRIBUTE ::= {
   { WITH SYNTAX ContentType             ID id-contentType         } |
   { WITH SYNTAX MessageDigest           ID id-messageDigest       } |
   { WITH SYNTAX SignaturePurposes       ID id-signaturePurpose    } |
   { WITH SYNTAX SigningTime             ID id-signingTime         } |
   { WITH SYNTAX SigningCertificate      ID id-signingCertificate } |
   { WITH SYNTAX OtherSigningCertificate ID id-otherSigningCert    } |
   { WITH SYNTAX BiometricSyntax         ID id-biometricSyntax     } |
```

```
   { WITH SYNTAX MsgSequenceNo           ID id-msgSequenceNo      } |
   { WITH SYNTAX Content                 ID id-contentIdentifier  } |
   { WITH SYNTAX MessageComponents       ID id-messageComponents  },

   ...  -- Expect additional objects --
}
```

6.2.2.1 Content Type

The content-type attribute identifies the type of content being signed. This attribute must be included in the signature computation when the content being signed is not ordinary data, or when any signed attributes are included in the message. This requirement allows malicious substitution of the **contentType** component of **EncapsulatedContentInfo** to be detected by the message recipient.

The <type> element of a content-type attribute contains the object identifier value **id-contentType**, which identifies one or more values of type **ContentType**. The **id-contentType** value is defined as:

```
id-contentType ::= <OID> 1.2.840.113549.1.9.3 </OID>
```

Type **ContentType** is defined in terms of the **&id** field of the **CONTENTS** information object set:

```
ContentType ::= CONTENTS.&id({Contents})

CONTENTS ::= TYPE-IDENTIFIER  -- Defined in ISO/IEC 8824-2, Annex A
```

A value of **ContentType** is a unique object identifier from the information object set **Contents**. The **Contents** information object set imposes a constraint on the valid values of **ContentType**. This set of objects is defined as:

```
Contents CONTENTS ::= {
   { ESignedData              IDENTIFIED BY  id-signedData            } |
   { EEnvelopedData           IDENTIFIED BY  id-envelopedData         } |
   { EAuthenticatedData       IDENTIFIED BY  id-ct-authData           } |
   { EDigestedData            IDENTIFIED BY  id-digestedData          } |
   { EEncryptedData           IDENTIFIED BY  id-encryptedData         } |
   { ENamedKeyEncryptedData   IDENTIFIED BY  id-namedkeyencryptedData } |
   { EData                    IDENTIFIED BY  id-data                  },
}
```

The valid values of type **ContentType** are the object identifiers **id-signedData**, **id-envelopedData**, **id-ct-authData**, **id-digestedData**, **id-encryptedData**, **id-namedkeyencryptedData**, and **id-data**.

In a value of type **SignerInfo**, a **signedAttrs** component containing a content-type attribute can be encoded using XML markup as:

```
<signedAttrs>
   <SignedAttribute>
      <type> 1.2.840.113549.1.9.3 </type>
      <values>
         <SET>
            <ContentType>
               1.2.840.113549.1.7.5
            </ContentType>
         </SET>
```

```
      </values>
   </SignedAttribute>
</signedAttrs>
```

Here the <signedAttrs> element contains a single attribute, a value of type **SignedAttribute**. The <type> element of <SignedAttribute> contains an **id-contentype** value that indicates this attribute is a content-type attribute. The <values> element contains a set of one value of type **ContentType**. The **ContentType** value in this example identifies the content as **DigestedData**.

6.2.2.2 Message Digest

The message-digest attribute carries a hash of the message being signed so that the message is indirectly included in the signature computation. This hash is a value of type **MessageDigest**. This attribute is required if any other signed attributes are present.

The <type> element of a message-digest attribute contains the object identifier value **id-messageDigest**, which identifies one or more values of type **MessageDigest**, an octet string. The **id-messageDigest** value is defined as:

id-messageDigest ::= <OID> 1.2.840.113549.1.9.4 </OID>

Type **MessageDigest** is defined as:

MessageDigest ::= OCTET STRING

In a value of type **SignerInfo**, a **signedAttrs** component containing a message-digest attribute can be encoded using XML markup as:

```
<signedAttrs>
   <SignedAttribute>
      <type> 1.2.840.113549.1.9.4 </type>
      <values>
         <SET>
            <MessageDigest>
               1DCA28DF401F13D5A49A17505DB229E401EE87C2
            </MessageDigest>
         </SET>
      </values>
   </SignedAttribute>
</signedAttrs>
```

Here the <signedAttrs> element contains a single attribute, a value of type **SignedAttribute**. The <type> element of <SignedAttribute> contains an **id-messageDigest** value that indicates this attribute is a message-digest attribute. The <values> element contains a set of one value of type **MessageDigest**.

6.2.2.3 Signature Purpose

The signature-purpose attribute may be used to indicate the reason a signature was applied. For example, the attribute might indicate the signature is to attach a time-stamp, to provide a receipt, etc. It is a set of values of type **OBJECT IDENTIFIER**, defined as type **SignaturePurposes**. ANS X9.45, Section 6.1.2 gives more information.

The `<type>` element of a signature-purpose attribute contains the object identifier value **`id-signaturPurpose`**, which identifies one or more values of type **`SignaturePurposes`**. The **`id-signaturePurpose`** value is defined as:

```
id-signaturePurpose ::= <OID> 1.2.840.10052.23 </OID>
```

Type **`SignaturePurposes`** is a series of zero or more values of type **`SignaturePurpose`**. This type is defined as:

```
SignaturePurposes ::= SEQUENCE SIZE(0..MAX) OF SignaturePurpose
```

Type **`SignaturePurpose`** is defined in terms of the **`&id`** field of the **`PURPOSE`** information object set:

```
SignaturePurpose ::= PURPOSE.&id({SignerPurposes})

PURPOSE ::= CLASS {
   &id  OBJECT IDENTIFIER  UNIQUE
}
  WITH SYNTAX { SIGNER &id }
```

A value of this type is a unique object identifier from the information object set **`SignerPurposes`**, which imposes a constraint on the valid values of type **`SignaturePurpose`**. The **`SignerPurposes`** information object set is defined as:

```
SignerPurposes PURPOSE ::= {
   { SIGNER id-authorization }  |
   { SIGNER id-cosignature   }  |
   { SIGNER id-witness       }  |
   { SIGNER id-receipt       }  |
   { SIGNER id-confirmation  }  |
   { SIGNER id-timestamp     }  |
   { SIGNER id-device        }  |
   { SIGNER id-registry      }  |
   { SIGNER id-integrity     },

   ...  -- Expect additional objects --
}
```

Each SignaturePurposes object is an object identifier defined as follows. These signature purposes are discussed further in ANS X9.45.

```
id-authorization  ::= <OID> 1.2.840.10052.1.23.1 </OID>
id-cosignature    ::= <OID> 1.2.840.10052.1.23.2 </OID>
id-witness        ::= <OID> 1.2.840.10052.1.23.3 </OID>
id-receipt        ::= <OID> 1.2.840.10052.1.23.4 </OID>
id-confirmation   ::= <OID> 1.2.840.10052.1.23.5 </OID>
id-timestamp      ::= <OID> 1.2.840.10052.1.23.6 </OID>
id-device         ::= <OID> 1.2.840.10052.1.23.7 </OID>
id-registry       ::= <OID> 1.2.840.10052.1.23.8 </OID>
id-integrity      ::= <OID> 1.2.840.10052.1.23.9 </OID>
```

In a value of type **`SignerInfo`**, a **`signedAttrs`** component containing a signature-purpose attribute can be encoded using XML markup as:

```
<signedAttrs>
   <SignedAttribute>
      <type> 1.2.840.10052.23 </type>
      <values>
         <SET>
            <SignaturePurposes>
               <SignaturePurpose>
                  1.2.840.10052.1.23.9
               </SignaturePurpose>
               <SignaturePurpose>
                  1.2.840.10052.1.23.4
               </SignaturePurpose>
            </SignaturePurposes>
            <SignaturePurposes>
               <SignaturePurpose>
                  1.2.840.10052.1.23.6
               </SignaturePurpose>
            </SignaturePurposes>
         </SET>
      </values>
   </SignedAttribute>
</signedAttrs>
```

Here the `<signedAttrs>` element contains a single attribute, a value of type **SignedAttribute**. The `<type>` element of `<SignedAttribute>` contains an **id-signaturePurpose** value that indicates this attribute is a signature-purpose attribute. The `<values>` element in this example contains a set of two values of type **SignaturePurposes**. Type **SignaturePurposes** is itself a series of one or more values of type **SignaturePurpose**, a signature purpose object identifier.

The first set of signature purposes contains two <SignaturePurpose> elements, indicating integrity and receipt purposes. The second set of signature purposes contains a single <SignaturePurpose> element. This element indicates a signature purpose of time stamping.

6.2.2.4 Signing Time

The signing-time attribute type may be used to attach signed date and time information to a message, to indicate that the message was created prior to that time. No requirement is imposed by this standard as to the correctness of the signing time, and the acceptance of a purported signing time is a matter of a recipient's discretion. It is expected, however, that some signers, such as time-stamp servers, will be trusted implicitly.

The `<type>` element of a signing-time attribute contains the object identifier value **id-signingTime**, which identifies one or more values of type **SigningTime**. The **id-signingTime** value is defined as:

```
id-signingTime ::= <OID> 1.2.840.113549.1.9.5 </OID>
```

Type **SigningTime** is defined as:

```
SigningTime ::= CHOICE {
   utcTime           UTCTime,
   generalizedTime   GeneralizedTime
}
```

In a value of type **SignerInfo**, a **signedAttrs** component containing a signing-time attribute can be encoded using XML markup as:

```
<signedAttrs>
   <SignedAttribute>
      <type> 1.2.840.113549.1.9.5 </type>
      <values>
         <SET>
            <SigningTime>
               <generalizedTime>
                  19801004000000Z
               </generalizedTime>
            </SigningTime>
         </SET>
      </values>
   </SignedAttribute>
</signedAttrs>
```

Here the <signedAttrs> element contains a single attribute, a value of type **SignedAttribute**. The <type> element of <SignedAttribute> contains an **id-signingTime** value that indicates this attribute is a signing-time attribute. The <values> element contains a set of one value of type **SigningTime**. The time string in this example represents midnight, October 4, 1980.

6.2.2.5 Signing Certificate

The signing-certificate attribute may be used to indicate the certificate required to verify a signature, and the policy under which the signature was applied. It has syntax:

The <type> element of a signing-certificate attribute contains the object identifier value **id-signingCertificate**, which identifies one or more values of type **SigningCertificate**. The **id-signingCertificate** value is defined as:

```
id-signingCertificate ::= <OID> 1.2.840.113549.1.9.16.2.12 </OID>
```

Type **SigningCertificate** is defined as:

```
SigningCertificate ::= SEQUENCE {
   certs     ESSCertIDs,
   policies  PolicyInfos  OPTIONAL
}

ESSCertIDs ::= SEQUENCE OF ESSCertID

ESSCertID ::=  SEQUENCE {
   certHash      Hash,
   issuerSerial  IssuerSerial  OPTIONAL
}

Hash ::= CHOICE {
   ietf       OCTET STRING,
   withAlgID  DigestInfo
}
```

In a value of type **SignerInfo**, a **signedAttrs** component containing a signing-certificate attribute can be encoded using XML markup as:

```
<signedAttrs>
   <SignedAttribute>
      <type> 1.2.840.113549.1.9.16.2.12 </type>
      <values>
         <SET>
            <SigningCertificate>
               <certs>
                  <ESSCertID>
                     <certHash>
                        <ietf>
                           1DCA28DF401F13D5A49A17505DB229E401EE87C2
                        </ietf>
                     </certHash>
                  </ESSCertID>
               </certs>
            </SigningCertificate>
         </SET>
      </values>
   </SignedAttribute>
</signedAttrs>
```

Here the `<signedAttrs>` element contains a single attribute, a value of type **SignedAttribute**. The `<type>` element of `<SignedAttribute>` contains an **id-signingCertificate** value that indicates this attribute is a signing-certificate attribute. The `<values>` element contains a set of one value of type **SigningCertificate**.

The `<certs>` element contains a single occurrence of the <ESSCertID> element, so that only a single signing certificate is identified. The optional `<policies>` element is not present in the parent `<SigningCertificate>` element. The presence of the `<ietf>` element indicates that the **ietf** choice alternative of type **Hash** is used to identify the signing certificate using a SHA-1 digest of the entire DER encoding of the certificate.

6.2.2.6 Other Signing Certificate

The other-signing-certificate attribute has syntax **OtherSigningCertificate**, and allows the use of any ANSI X9 approved cryptographic hash algorithm.

The `<type>` element of an other-signing-certificate attribute contains the object identifier value **id-otherSigningCert**, which identifies one or more values of type **OtherSigningCertificate**. The **id-otherSigningCert** value is defined as:

```
id-otherSigningCert ::= <OID> 0.4.0.1733.1.1.12 </OID>
```

Type **OtherSigningCertificate** is defined as:

```
OtherSigningCertificate ::= SEQUENCE {
   certs     OtherCertIDs,
   policies  PolicyInfos  OPTIONAL
}

OtherCertIDs ::= SEQUENCE OF OtherCertID
```

```
OtherCertID ::= SEQUENCE {
   certHash      Hash,
   issuerSerial  IssuerSerial  OPTIONAL
}
```

In a value of type **SignerInfo**, a **signedAttrs** component containing an other-signing-certificate attribute can be encoded using XML markup as:

```
<signedAttrs>
   <SignedAttribute>
      <type> 0.4.0.1733.1.1.12 </type>
      <values>
         <SET>
            <OtherSigningCertificate>
               <certs>
                  <OtherCertID>
                     <certHash>
                        <ietf>
                           3D5AF40149A175051DCA28DF1DB229E401C2EE87
                        </ietf>
                     </certHash>
                  </OtherCertID>
               </certs>
            </OtherSigningCertificate>
         </SET>
      </values>
   </SignedAttribute>
</signedAttrs>
```

Here the <signedAttrs> element contains a single attribute, a value of type **SignedAttribute**. The <type> element of <SignedAttribute> contains an **id-otherSigningCert** value that indicates this attribute is an other-signing-certificate attribute. The <values> element contains a set of one value of type **OtherSigningCertificate**.

The <certs> element contains a single occurrence of the <OtherCertID> element, so that only a single certificate is identified. The optional <policies> element is not present in the parent <OtherSigningCertificate> element. The presence of the <ietf> element indicates that the **ietf** choice alternative of type **Hash** is used to identify the other signing certificate using a SHA-1 digest of the entire DER encoding of the certificate.

6.2.2.7 Biometric Object

The biometric-object attribute is used to convey biometric information. This attribute has syntax **BiometricSyntax**, and is defined in ANS X9.84 [13]. The biometric data may already be signed or encrypted, in which case the information may be conveyed as an unsigned attribute.

The <type> element of a biometric-object attribute contains the object identifier value **id-biometricSyntax**, which identifies one or more values of type **BiometricSyntax**. The **id-biometricSyntax** value is defined as:

```
id-biometricSyntax ::= <OID> 1.2.840.10060.1.2 </OID>
```

In a value of type **SignerInfo**, a **signedAttrs** component containing a biometric-object attribute can be encoded using XML markup as:

```
<signedAttrs>
   <SignedAttribute>
      <type> 1.2.840.10060.1.2 </type>
      <values>
         <SET>
            <BiometricSyntax>
               ...
            </BiometricSyntax>
         </SET>
      </values>
   </SignedAttribute>
</signedAttrs>
```

Here the <signedAttrs> element contains a single attribute, a value of type **SignedAttribute**. The <type> element of <SignedAttribute> contains an **id-biometricSyntax** value that indicates this attribute is a biometric-object attribute. The <values> element contains a set of one value of type **BiometricSyntax**.

6.2.2.8 Sequence Number

The sequence-number attribute may be used by the application to maintain pair wise counters or sequence numbers between two entities. This sequence number is a value of type **MsgSequenceNo** and is defined in this standard as an integer greater than zero.

The <type> element of a sequence-number attribute contains the object identifier value **id-msgSequenceNo**, which identifies one or more values of type **MsgSequenceNo**. The **id-msgSequenceNo** value is defined as:

```
id-msgSequenceNo ::= <OID> 1.2.840.10060.1.1 </OID>
```

Type **MsgSequenceNo** is defined as:

```
MsgSequenceNo ::= INTEGER (0..MAX)
```

In a value of type **SignerInfo**, a **signedAttrs** component containing a sequence-number attribute can be encoded using XML markup as:

```
<signedAttrs>
   <SignedAttribute>
      <type> 1.2.840.10060.1.1 </type>
      <values>
         <SET>
            <MsgSequenceNo>
               68
            </MsgSequenceNo>
         </SET>
      </values>
   </SignedAttribute>
</signedAttrs>
```

Here the <signedAttrs> element contains a single attribute, a value of type **SignedAttribute**. The <type> element of <SignedAttribute> contains an **id-msgSequenceNo** value that indicates this attribute is a sequence-number attribute. The <values> element contains a set of one value of type **MsgSequenceNo**.

6.2.2.9 Content Identifier

The content-identifier attribute may be used by an application to attach a meaningful identifier to the message. A standard means of message identification is not defined, and is left to the application. A nonce, a URI, a date and time string, a string of descriptive text, or some other indication may be used.

The `<type>` element of a content-identifier attribute contains the object identifier value **`id-contentIdentifier`**, which identifies one or more values of type **`Content`**. The **`id-contentIdentifier`** value is defined as:

```
id-contentIdentifier ::= <OID> 1.2.840.113549.1.9.16.2.7 </OID>
```

Type **`Content`** is defined as:

```
Content ::= OCTET STRING
```

In a value of type **`SignerInfo`**, a **`signedAttrs`** component containing a content-identifier attribute can be encoded using XML markup as:

```
<signedAttrs>
   <SignedAttribute>
      <type> 1.2.840.113549.1.9.16.2.7 </type>
      <values>
         <SET>
            <Content>
               1066
            </Content>
         </SET>
      </values>
   </SignedAttribute>
</signedAttrs>
```

Here the `<signedAttrs>` element contains a single attribute, a value of type **`SignedAttribute`**. The `<type>` element of `<SignedAttribute>` contains an **`id-contentIdentifier`** value that indicates this attribute is a content-identifier attribute. The `<values>` element contains a set of one value of type **`Content`**.

6.2.2.10 Message Components

The message-components attribute carries a list of those parts of a message that are being signed. The message can be any ASN.1 defined type. The list of message components may be in any order chosen by the signer. The complete message from which the list of parts is derived is the value of the optional **`eContent`** component of type **`EncapsulatedContentInfo`**.

The `<type>` element of a message-components attribute contains the object identifier value defined in this standard, **`id-messageComponents`**. The **`id-messageComponents`** value is defined as:

```
id-messageComponents ::= <OID> 1.3.133.16.840.9.96.1.1 </OID>
```

This OID identifies one or more values of type **`MessageComponents`**, a sequence of values of type **`Component`**, which is defined as type **`UTF8String`**. Type **`MessageComponents`** is defined as:

```
MessageComponents ::= SEQUENCE SIZE(1..MAX) OF Component
```

```
Component ::= UTF8String
```

Each value of type **`Component`** identifies one fully qualified element in an ASN.1 module. Each element is a dotted string of characters of the form **`Module.Type[.index][.identifier],`** where

`Module` The name of an ASN.1 module. An ASN.1 module name appears as the first symbol in the first line of the module. Within a given set of ASN.1 modules used in an encoding application, each module is required to have a unique module identifier name. This allows references to defined types having the same name but existing in more than one module to be uniquely identified.

`Type` The name of any ASN.1 type defined in **`Module`**. The first symbol of an assignment that defines a type or a parameterized type uniquely identifies that type within a given ASN.1 module.

`index` An optional positive integer value indicating the specific instance of a type, or the optional wildcard symbol "**`*`**" indicating all such instances, in a **`SEQUENCE OF`** or **`SET OF`** type. The value "**`1`**" indicates the first instance of the type. When the **`index`** is not present and the **`Type`** is a **`SEQUENCE OF`** or **`SET OF`** type, the wildcard symbol is assumed.

`identifier` The optional unique name of a component of the ASN.1 type identified by **`Module.Type`**.

Note that in the example:

```
P DEFINITIONS ::= BEGIN
   G ::= SEQUENCE {
      one  INTEGER,
      too  SET OF PrintableString
   }
END
```

"**`P`**" is a **`Module`** name, "**`G`**" is a **`Type`** name, and "**`one`**" and "**`too`**" are the "**`identifier`**" names of the two components of type **`G`**. The notation "**`P.G.one`**" may be used in a value of type **`Component`** to indicate that the complete encoding of a value of type **`INTEGER`** is to be signed. The notation "**`P.G.too.2`**" indicates the second instance in a set of values of type **`PrintableString`**. The notations "**`P.G.too`**" and "**`P.G.too.*`**" both indicate the entire set of values.

Using this example ASN.1 module **`P`**, and notation to indicate that all instances in the set of values of type **`PrintableString`** are to be signed, a value of type **`SignerInfo`**, a **`signedAttrs`** component containing a message-components attribute can be encoded using XML markup as:

```
<signedAttrs>
   <SignedAttribute>
      <type> 1.3.133.16.840.9.96.1.1 </type>
      <values>
         <SET>
            <MessageComponents>
               <Component> P.G.too </Component>
            </MessageComponents>
         </SET>
```

```
      </values>
   </SignedAttribute>
</signedAttrs>
```

Here the <signedAttrs> element contains a single attribute, a value of type **SignedAttribute**. The <type> element of <SignedAttribute> contains an **id-messageComponents** value that indicates this attribute is a message-components attribute. The <values> element of <SignedAttribute> contains a set of one value of type **MessageComponents**, which indicates the list of message components to be signed, and which contains a single value of type **Component**.

The notation <Component> **P.G.too** </Component> indicates that the complete encoding of a value of type **SET OF INTEGER**, including the starting and ending XML markup tags <too> and </too>, are input to the message digest phase of the digital signature process.

When more than one value of type **Component** is present in the list, the input to the message digest processing is the concatenation, in the order presented in the list, of a series of complete encodings of the values indicated. Though the content to be signed may be detached or included in the optional **eContent** component of type **EncapsulatedContentInfo**, it is an error if a value of type **Component** indicates a value that is not present in the content to be signed.

6.2.3 Unsigned Attributes

This section defines two useful attributes that are not signed. One of these, the biometric-object attribute, as an option, may also appear in using applications as a signed attribute. Applications may define their own attributes as well.

```
UnsignedAttributes ::= SET SIZE(1..MAX) OF UnsignedAttribute

UnsignedAttribute ::= Attribute {{Unsigned}}

Unsigned ATTRIBUTE ::= {
   { WITH SYNTAX Countersignature ID id-countersignature } |
   { WITH SYNTAX BiometricSyntax  ID id-biometricSyntax  },

   ...  -- Expect additional objects --
}
```

6.2.3.1 Counter Signature

The countersignature attribute carries a signature computed over the **signature** component of the value of type **SignerInfo** in which this countersignature appears as an attribute.

The <type> element of a content-type attribute contains the object identifier value **id-countersignature**, which identifies one or more values of type **Countersignature**. The **id-countersignature** value is defined as:

```
id-countersignature ::= <OID> 1.2.840.113549.1.9.6 </OID>
```

Type **Countersignature** is defined as:

```
Countersignature ::= SignerInfo
```

Countersignatures can provide proof of the order of signature application. Countersignature values have the same requirements as values of type **SignerInfo** in **SignedData** for ordinary signatures, except that:

— The **signedAttrs** component must contain a message-digest attribute if it contains any other attributes, but need not contain a content-type attribute;

— The input to the message-digesting process is the value octets (not including the tag and link octets) of the DER encoding of the **signature** component of the **SignerInfo** value with which the attribute is associated.

In a value of type **SignerInfo**, a **signedAttrs** component containing a message-digest attribute can be encoded using XML markup as:

```
<unsignedAttrs>
   <UnsignedAttribute>
      <type> 1.2.840.113549.1.9.6 </type>
      <values>
         <SET>
            <Countersignature>
               <version> 96 </version>
               <sid>
                  <certHash>
                  <ietf>
                     401F1E87C29E23D1DCA28D17F5A49A505DB2401E
                  </ietf>
                  </certHash>
               </sid>
               <digestAlgorithm>
                  <algorithm> 1.3.14.3.2.26 </algorithm>
                  <parameters> <NullParms/> </parameters>
               </digestAlgorithm>
               <signatureAlgorithm>
                  <algorithm> 1.2.840.113549.1.1.5 </algorithm>
                  <parameters> <NullParms/> </parameters>
               </signatureAlgorithm>
               <signature>
                  A28D17F5A49A5 ... BAD21C712F854BA5
               </signature>
            </Countersignature>
         </SET>
      </values>
   </UnsignedAttribute>
</unsignedAttrs>
```

Here the <unsignedAttrs> element contains a single attribute, a value of type **CounterSignature**. The <type> element of <UnsignedAttribute> contains an object identifier that indicates this attribute is a countersignature attribute. The <values> element contains a set of one value of type **CounterSignature**.

6.2.3.2 Biometric Object

The biometric-object attribute is used to convey biometric information. This attribute has syntax **BiometricSyntax**, and is defined in ANS X9.84. The biometric data may already be signed or encrypted, in which case the information may be conveyed as an unsigned attribute.

```
id-biometricSyntax ::= <OID> 1.2.840.10060.1.2 </OID>
```

In a value of type **SignerInfo**, a **signedAttrs** component containing a biometric-object attribute can be encoded using XML markup as:

```
<unsignedAttrs>
   <UnsignedAttribute>
      <type> 1.2.840.113549.1.9.4 </type>
      <values>
         <SET>
            <BiometricObject>
               <biometricHeader>
                  <version> 0 </version>
                  <recordType> <id> 5 </id> </recordType>
                   <dataType> <processed/> </dataType>
                  <purpose> <audit/> </purpose>
                  <quality> -1 </quality>
                  <validityPeriod>
                     <notBefore> 1986.7.13 </notBefore>
                     <notAfter> 2003.7.12.23.59.59 </notAfter>
                  </validityPeriod>
                  <format>
                     <formatOwner>
                        <oid> 2.23.42.9.10.4.2.2 </oid>
                     </formatOwner>
                  </format>
               </biometricHeader>
               <biometricData> A239C ... DE0B2C1D </biometricData>
            </BiometricObject>
         </SET>
         </values>
   </UnsignedAttribute>
</unsignedAttrs>
```

Here the `<signedAttrs>` element contains a single attribute, a value of type **SignedAttribute**. The `<type>` element contains an object identifier that indicates this attribute is a biometric-object attribute. The `<values>` element contains a set of one value of type **BiometricObject**.

6.2.4 Certificate Formats

This standard supports all of the certificate formats defined ANS X9.73, including X.509 version three certificates, version two attribute certificates, and the compact domain certificate format defined in ANS X9.68.

6.2.5 Detached Signatures

Detached signatures are signatures that are conveyed separately from the content in an **EncapsulatedContentInfo** value. As with the other data types defined here, the content, i.e. the **eContent** component of the **EncapsulatedContentInfo** type, is optional. This allows the content to be conveyed separately, with the application maintaining the connection between the content and the signature(s).

For example, applications can convey the content as one MIME body part, and the signature(s) as another. This allows a recipient to process the content while ignoring the signature body part if the application is not capable of signature verification.

6.2.6 Signature Process

A message digest is used to create the digital signature carried in the **signature** component of the **SignerInfo** component of type **SignedData**. The message digest is calculated using the **algorithm** and **parameters** components of a value of type **DigestAlgorithmIdentifier** indicated by the **digestAlgorithm** component of **SignerInfo**, the value of the **eContent** component of type **EncapsulatedContentInfo**, and any attributes in the **signedAttrs** component of **SignerInfo**. The **eContentType** component of **EncapsulatedContentInfo** identifies the type of message content being signed.

When a value of type **SignedData** is represented as XML markup, the starting and ending **eContent** tags are excluded from the message digest process. Only the "value" portion of the complete canonical XER encoding of **eContent** is digested. This can be any sort of data whatsoever, including the Base64 armored contents of a word processing file, fragments of an XML document, or an XML schema. The `<eContent>` and `</eContent>` tags of a value of type **EncapsulatedContentInfo** are excluded from the message digest process.

6.2.6.1 Ordinary Data

When the content type is ordinary data and there are no attributes to be signed, the value of the **eContent** component of **EncapsulatedContentInfo** is digested. The result of the message digest process is then digitally signed using the signer's private key and the signature **algorithm** and **parameters** specified in the **signatureAlgorithm** component of type **SignerInfo**. The result of the signature process becomes the value of the **signature** component of the **SignerInfo** component of type **SignedData**.

6.2.6.2 Authenticated Attributes

When the content type is not ordinary data or when there are any attributes to be signed, the message digest must be computed on the content being signed together with the authenticated attributes. The initial input to the message digest process is the value of the **eContent** component of **EncapsulatedContentInfo**. Only the value of the **eContent** component is digested, and the `<eContent>` and `</eContent>` tags of a value of type **EncapsulatedContentInfo** are excluded from the message digest process.

When the optional signedAttrs component of type SignedData is present, … TBS

6.3 Authenticated Data

The authenticated-data message consists of message content and an ANSI X9 approved symmetric key authentication code [1], along with key management information used to convey the verification key to the recipients. Data integrity is provided by use of this type. Origin authentication may also be provided when an appropriate key management technique such as key agreement is used by the message originator.

```
AuthenticatedData ::= SEQUENCE {
   version                    Version,
   originatorInfo             [0] OriginatorInfo  OPTIONAL,
   recipientInfos             RecipientInfos,
   macAlgorithm               MACAlgorithmIdentifier,
   digestAlgorithm            [1] DigestAlgorithmIdentifier OPTIONAL,
   encapContentInfo           EncapsulatedContentInfo,
   authenticatedAttributes    [2] AuthAttributes  OPTIONAL,
   mac                        MessageAuthenticationCode,
   unauthenticatedAttributes  [3] UnauthAttributes  OPTIONAL
}
```


The value of **version** is the schema version number. This value shall be ninety-six for this standard.

The **originatorInfo** component provides public key certificate and certificate revocation information about the message originator. It is present only if required by the key management method. It may contain certificates and CRLs that have been. Base64 armored to minimize their size when represented using XML markup, while preserving their original ASN.1 BER encodings. The Base64 processing of these values is fully described in section 6.2.1. The constraints on values of type **OriginatorInfo** specified in Section 6.7.2 Certificate Formats, also apply to the use of **AuthenticatedData** with domain certificates.

The **recipientInfos** component contains a list of per-recipient information. There shall be at least one element in the list of values of type **RecipientInfo**. All constraints on **RecipientInfo** defined in Section 6.7.2, Certificate Formats, also apply to the use of **AuthenticatedData** with domain certificates.

The **macAlgorithm** component identifies an X9 approved message authentication code algorithm (a MAC or an HMAC [11] algorithm) used by the message sender, and any associated algorithm parameters. The type of the values in this component is **MACAlgorithmIdentifier**, which is defined as:

```
MACAlgorithmIdentifier::= AlgorithmIdentifier {{MACAlgorithms}}

MACAlgorithms ALGORITHM ::= {
   { OID hmac-with-SHA1 },

   ...  -- expect other MAC or HMAC algorithms --
}
```

The **digestAlgorithm** component identifies an X9 approved message digest algorithm used by the message sender, and any associated algorithm parameters. This component is optional, but must be present when there are any authenticated attributes. X9 approved digest algorithms are fully specified in section 6.2.1.

The **encapContentInfo** component is a value of type **EncapsulatedContentInfo**, which is defined in section 6.1, identifies and optionally carries the authenticated message content. The **eContentType** component of type **EncapsulatedContentInfo** is an object identifier value that indicates the content type. When present, the optional **eContent** component of type **EncapsulatedContentInfo** contains the message content. This component may be absent to allow construction of “detached authentication codes”, but when the **eContent** value is absent, the message sender calculates the authentication code on the message content as though the value were present.

The **authenticatedAttributes** component is a collection of attributes that are authenticated along with the message content. Some useful authenticated attributes are defined in section 6.2.2.

The **mac** component contains the results of calculating a message authentication code on the message content, and is defined as:

```
MessageAuthenticationCode ::= OCTET STRING
```

The **unauthenticatedAttributes** component is a collection of attributes that are not authenticated along with the message content. Some useful attributes that are not authenticated are defined in section 6.2.3.

The **AuthenticatedData** cryptographic content type is indicated by the object identifier value:

```
id-ct-authData ::= <OID> 1.2.840.113549.1.9.16.1.2 </OID>
```

A value of type **AuthenticatedData** can be encoded using XML markup as:

```
<AuthenticatedData>
   <version> 96 </version>
   <recipientInfos>
      <RecipientInfo>
         <ktri>
            <version> 96 </version>
            <rid>
               <rKeyId>
                  <subjectKeyIdentifier>
                     1DC17505DB229E401EE87C2A28DF401F13D5A49A
                  </subjectKeyIdentifier>
               </rKeyId>
            </rid>
            <keyEncryptionAlgorithm>
               <algorithm> 1.2.840.113549.1.1.1 </algorithm>
               <parameters> </NullParms> </parameters>
            </keyEncryptionAlgorithm>
            <encryptedKey>
               229E402819B301DD9127488201F3A16C24BF7CCA482B
            </encryptedKey>
         </ktri>
      </RecipientInfo>
   </recipientInfos>
   <macAlgorithm>
      <algorithm> 1.3.6.1.5.5.8.1.2 </algorithm>
    </macAlgorithm>
   <encapContentInfo>
      <eContentType> 1.2.840.113549.1.7.1 </eContentType>
   </encapContentInfo>
   <mac>
      1DC17505DB229E401EE87C2A28DF401F13D5A49A17505DB229E401EE87C2
   </mac>
</AuthenticatedData >
```

Here the value in the <version> element identifies this value of type **AuthenticatedData** as conforming to this standard. The <recipientInfos> element contains only one element, indicating that there is only one recipient of the information. The <ktri> element identifies the key transport key management technique being used for this message recipient, and the <algorithm> element indicates that the key encryption algorithm used to encrypt the symmetric key for transport is **rsaEncryption**. The <macAlgorithm> element indicates that an HMAC algorithm was used to create the value in the <mac> element. The <eContentType> element identifies the MACed content as ordinary data.

6.3.1 MAC and HMAC Creation

A message authentication code (MAC or HMAC [11]) may be calculated on a message content value of any type, or calculated on a digest of the message content together with a collection of one or more authenticated attributes. The calculation process uses the X9 approved authentication code algorithm, and any associated algorithm parameters, indicated in the **macAlgorithm** component of type **AuthenticatedData**. The process also uses the message content, together with any optional authenticated attributes, and the authentication key conveyed in a value of type **RecipientInfo** for the message recipient. This key may be prearranged, or chosen at random, but shall be generated in accordance to the requirements of the X9 approved algorithm being used.

The result of this calculation becomes the message authentication code in the **mac** component of type **AuthenticatedData**. If the optional **authenticatedAttributes** component of type **AuthenticatedData** is not present, just the value of the **eContent** component of the **encapContentInfo** component of type **AuthenticatedData** is input to the message authentication code calculation process. The <eContent> and </eContent> elements are not included in the input.[3]

If the optional **authenticatedAttributes** component is present in a value of type **AuthenticatedData**, then the authenticated attributes are the input to the message authentication code calculation process, and both the **`contentType`** and **`messageDigest`** attributes defined in section 6.2 must be included in the input. The **`contentType`** attribute shall indicate the type of message content being authenticated. The **`messageDigest`** attribute shall include a message digest of that content. The `<authenticatedAttributes>` and `</authenticatedAttributes>` elements that encapsulate the collection of authenticated attributes shall also be included in the input to the calculation process.

The value of the **authenticatedAttributes** component must be encoded using the canonical variant (cXER) of the XML Encoding Rules (XER), even though the rest of the cryptographic message may be encoded using basic XER. The **authenticatedAttributes** component shall be present when the content is not ordinary data identified by **`id-data`**. When any authenticated attributes are present, the optional **`digestAlgorithm`** component of type **`AuthenticatedData`** shall also be present to indicate the digest algorithm used to create the **`messageDigest`** attribute. The input to the message digest process shall be the value of the **eContent** component of the **encapContentInfo** component of type **AuthenticatedData**. The <eContent> and </eContent> elements are not included in the input.

6.3.2 MAC and HMAC Verification

To verify a message authentication code in the **`mac`** component of a value of type **`AuthenticatedData`**, a MAC or HMAC is computed using the same key used by the sender, the key carried in the RecipientInfo value for the message recipient. If there are no authenticated attributes present, a message authentication code is calculated on the message content, the value in the `<eContent>` element. The resulting value is compared to the value of the **mac** component of type **`AuthenticatedData`**. If they are identical, the value in the **mac** component is valid.

If authenticated attributes are present, a digest is computed on the message content, the value in the `<eContent>` element. For the message digest process to succeed, this resulting value must be equivalent to the value sent by the message originator in the **`messageDigest`** attribute. Next, a message authentication code is calculated on the authenticated attributes. The resulting value is compared to the value of the **mac** component of type **`AuthenticatedData`**. If they are identical, the value in the **mac** component is valid.

6.4 Digested Data

A digested-data message consists of a message content identifier and a cryptographic hash of the identified content. The message content may also be present. Frequently, this message type is used as a building block for the creation of other messages, and the message content typically will be absent in the digested-data message, since this content can be carried elsewhere in the overall message.

---

[3] This processing requirement matches the requirement specified for use with binary encodings of **`eContent`** in both ANS X9.73 and ANS X9.84. In these standards, the tag and length octets of **`eContent`** are excluded from the MAC and HMAC calculation process, so that the length of the content being authenticated need not be known when the process is initiated. While not necessary for XML markup, which is always indefinite length encoded, this requirement allows a single programming solution to be used to implement ANS X9.73, ANS X9.84, and this standard.


```
DigestedData ::= SEQUENCE {
   version           Version,
   digestAlgorithm   DigestAlgorithmIdentifier,
   encapContentInfo  EncapsulatedContentInfo,
   digest            Digest
}
```

The value of **version** is the schema version number. This value shall be ninety-six for this standard.

The **digestAlgorithm** component identifies an X9 approved message digest algorithm used by the message sender, and any associated algorithm parameters.

The **encapContentInfo** component is a value of type **EncapsulatedContentInfo**, which is defined in section 6.1.

The **digest** component is a value of type **Digest**. This value contains the result of applying the hash function indicated by the **digestAlgorithm** component to the message content. Type **Digest** is defined as:

```
Digest ::= OCTET STRING
```

The **DigestedData** cryptographic content type is indicated by the object identifier value:

```
id-digestedData ::= <OID> 1.2.840.113549.1.7.5 </OID>
```

A value of type **DigestedData** can be encoded using XML markup as:

```
<DigestedData>
   <version> 96 </version>
   <digestAlgorithm>
      <algorithm> 2.16.840.1.101.3.4.2.1 </algorithm>
    </digestAlgorithm>
   <encapContentInfo>
      <eContentType> 1.2.840.113549.1.7.1 </eContentType>
   </encapContentInfo>
   <digest>
      1DC17505DB229E401EE87C2A28DF401F13D5A49A17505DB229E401EE87C2
   </digest>
</DigestedData>
```

Here the value in the <version> element identifies this value of type **DigestedData** as conforming to this standard. The <digestAlgorithm> element indicates that the digest algorithm used to create the value in the <digest> element is a two hundred and fifty-six bit FIPS 180-2 Secure Hash Algorithm. The <eContentType> element identifies the digested content as ordinary data.

6.5 Encrypted Data

An encrypted-data message consists of encrypted message content without any associated key management information. It is typically used as a building block in other messages, or for local protected data storage.

```
EncryptedData ::= SEQUENCE {
   version                Version,
   encryptedContentInfo   EncryptedContentInfo
}
```


The value of **version** is the schema version number. This value shall be ninety-six for this standard.

The **encryptedContentInfo** component is a value of type **EncryptedContentInfo** defined as:

```
EncryptedContentInfo ::= SEQUENCE {
   contentType                  ContentType,
   contentEncryptionAlgorithm   ContentEncryptionAlgorithmIdentifier,
   encryptedContent             [0] EncryptedContent  OPTIONAL
}
```

The type of encrypted content is indicated by an object identifier value in the **contentType** component.

The **contentEncryptionAlgorithm** component identifies content encryption algorithm and any associated algorithm parameters used to encrypt the message content.

The **encryptedContent** component contains the result of encrypting the content identified by the **contentType** component using the algorithm and parameters identified in the **contentEncryptionAlgorithm** component.

The **EncryptedData** cryptographic content type is indicated by the object identifier value:

```
id-encryptedData ::= <OID> 1.2.840.113549.1.7.6 </OID>
```

Examples of encrypted-data use in financial services include POS transactions, ATM transactions and encrypted PINs, where the content encryption key is communicated by other means, perhaps by wrapping this message within a signed-data message that includes a key identification attribute.

A value of type **EncryptedData** can be encoded using XML markup as:

```
<EncryptedData>
   <version> 96 </version>
   <encryptedContentInfo>
      <contentType> 1.2.840.113549.1.7.5 </contentType>
      <contentEncryptionAlgorithm>
         <algorithm> 1.2.840.113549.3.7 </algorithm>
         <parameters>
            <IV> 14FEA1DE6E3BC59C </IV>
         </parameters>
      </contentEncryptionAlgorithm>
      <encryptedContent>
         4A915A2D ... EAF8732B
      </encryptedContent>
   </encryptedContentInfo>
</EncryptedData>
```

Here the value in the `<version>` element identifies this value of type **EncryptedData** as conforming to this standard. The `<contentType>` element identifies the encrypted content as digested-data, a nested cryptographic type defined in this standard. The `<contentEncryptionAlgorithm>` identifies the Triple DES algorithm and its associated parameters, and initialization vector, `<IV>`.

6.6 Named Key Encrypted Data

A named-key-encrypted-data message consists of content encrypted with a single key and a key name. The name of the key is given in the **keyName** component of type **NamedKeyEncryptedData**. The encrypted content is carried in the **encryptedData** component, a value of type **EncryptedData**. Type **NamedKeyEncryptedData** is defined as:

```
NamedKeyEncryptedData ::= SEQUENCE {
   keyName        OCTET STRING (SIZE (1..MAX)),
   encryptedData  EncryptedData
 }
```

The **NamedKeyEncryptedData** cryptographic content type is indicated by the object identifier value:

```
id-namedkeyencryptedData ::= <OID> 1.2.840.10060.1.2 </OID>
```

Examples of named-key-encrypted-data use in financial services include POS transactions, ATM transactions and encrypted PINs, where the name of the key may be used to identify a particular device.

A value of type **NamedKeyEncryptedData** can be encoded using XML markup as:

```
<NamedKeyEncryptedData>
   <keyName> A6EF73B45ADEA73D1E </keyName>
   <encryptedData>
      <version> 96 </version>
      <encryptedContentInfo>
         <contentType> 1.2.840.113549.1.7.6 </contentType>
         <contentEncryptionAlgorithm>
            <algorithm> 1.2.840.113549.3.7 </algorithm>
            <parameters>
               <IV> 14FEA1DE6E3BC59C </IV>
            </parameters>
         </contentEncryptionAlgorithm>
         <encryptedContent>
            4A915A2D ... EAF8732B
         </encryptedContent>
      </encryptedContentInfo>
   </encryptedData>
</NamedKeyEncryptedData>
```

Here the value in the `<version>` element identifies this value of type **NamedKeyEncryptedData** as conforming to this standard. The `<keyName>` element contains the name of the key used to encrypt the value in the `<encryptedContent>` element, and the `<contentType>` element identifies the encrypted content as ordinary data. The `<contentEncryptionAlgorithm>` identifies the Triple DES algorithm and its associated parameters, and initialization vector, `<IV>`.

6.7 Enveloped Data

6.7.1 General

The enveloped-data message consists of an encrypted message content, along with key management information for each recipient. The content is encrypted under a symmetric content-encryption key (CEK). The

CEK is, in turn, encrypted under each recipient's public key or a shared symmetric key, and included in the key management information. Any type of content can be enveloped for any number of recipients.

The typical application of the enveloped-data content type will represent one or more recipients' digital envelopes on the content of the data or signed-data content types.

A recipient opens the envelope by decrypting one of the encrypted CEK and decrypting the encrypted message content with the recovered CEK.

The following object identifier identifies the enveloped-data content type:

```
id-envelopedData ::= <OID> 1.2.840.113549.1.7.3 </OID>
```

Type **EnvelopedData** is defined as:

```
EnvelopedData ::= SEQUENCE {
   version               Version,
   originatorInfo        [0] OriginatorInfo  OPTIONAL,
   recipientInfos        RecipientInfos,
   encryptedContentInfo  EncryptedContentInfo,
   unprotectedAttrs      [1] UnprotectedAttributes  OPTIONAL
}
```

The value of **version** is the schema version number. This value shall be ninety-six for this standard.

The **originatorInfo** component provides information about the originator. It is present only if required by the key management method. It may contain certificates and CRLs that have been Base64 armored to minimize their size when represented using XML markup, while preserving their original ASN.1 BER encodings. The Base64 processing of these values is fully described in section 6.2.1.

```
OriginatorInfo ::= SEQUENCE {
   certs  [0] CertificateSet  OPTIONAL,
   crls   [1] CertificateRevocationLists  OPTIONAL
}
```

certs is an optional collection of one or more certificates.

crls is an optional collection of one or more certificate revocation lists.

recipientInfos is a collection of per-recipient information. There shall be at least one element in the collection.

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

RecipientInfo ::= CHOICE {
   ktri   KeyTransRecipientInfo,
   kari   [1] KeyAgreeRecipientInfo,
   kekri  [2] KEKRecipientInfo,
   --  Choice [3] reserved for IETF password-based encryption mechanism
   ori    [4] ExtendedKeyMgmtRecipientInfo
}
```

Four key management techniques are supported in the choice alternatives of type **RecipientInfo**:

1) **`ktri`** - key transport: the content-encryption key is encrypted in the recipient's public key;

2) **`kari`** - key agreement: the recipient's public key and the sender's private key are used to generate a pair wise symmetric key, then the content-encryption key is encrypted in the pair wise symmetric key;

3) **`kekri`** - pre-established key encrypted keys: the content-encryption key is encrypted in a previously distributed symmetric key; and

4) **`ori`** - external mechanisms: this allows the use of additional key management mechanisms. In this standard, constructive key management, as defined in ANS X9.69 [10], is implemented as an external mechanism.

The **`RecipientInfo`** construct is extended in this standard to support these mechanisms.

Section 7, Key Management, discusses key management using ANSI X9 approved algorithms in detail.

`encryptedContentInfo` is the encrypted contents.

```
EncryptedContentInfo ::= SEQUENCE {
   contentType                  ContentType,
   contentEncryptionAlgorithm  ContentEncryptionAlgorithmIdentifier,
   encryptedContent            [0] EncryptedContent  OPTIONAL
}
```

The **`contentType`** indicates the type of content.

The **`contentEncryptionAlgorithm`** identifies the X9 approved content encryption algorithm [??], and any associated parameters, used to encrypt the content. The same content-encryption algorithm and content-encryption key is used for all recipients.

```
ContentEncryptionAlgorithmIdentifier ::=
                     AlgorithmIdentifier {{ContentEncryptionAlgorithms}}

ContentEncryptionAlgorithms ALGORITHM ::= {
   { OID des-ede3-cbc PARMS IV },

   ...  -- Expect other content encryption algorithms --
}
```

The **`encryptedContent`** is the result of encrypting the content. This component is optional and if not present in the message must be provided by other means. When the encrypted content is not present, it is the responsibility of the communicating applications to associate the encrypted content with the encryption key.

```
EncryptedContent ::= OCTET STRING
```

The **`unprotectedAttrs`** component of type **`EnvelopedData`** is a collection of attributes that are not protected by encryption.

```
UnprotectedAttributes ::= SET SIZE(1..MAX) OF Attribute {{ Unprotected }}

Unprotected ATTRIBUTE ::= { ...  -- Expect additional objects -- }
```

6.7.2 Certificate Formats

As discussed in Section 6.2.4, Certificate Formats, this standard supports the use of domain certificates as defined in ANS X9.68. This is reflected in the following areas of the **EnvelopedData** syntax that is defined in Section 6.7.1 above:

1) The certificates in **OriginatorInfo** may include domain certificates in type **CertificateSet**;

2) The **rid** component in **KeyTransRecipientInfo** and the **originatorCert** component in **KeyAgreeRecipientInfo** must contain subject key identifiers or certificate hashes rather than the issuer and serial number specified by a CA;

3) The **rid** component of **RecipientEncryptedKey** shall use the **certHash** choice alternative;

4) The domain certificate extensions may contain subject key identifiers. If not, the extensions are identified using the **certHash** choice alternative.

7 Key Management

7.1 General

This section defines mechanisms for conveying a symmetric key (for encryption or the computation of an authentication code) in a key management information structure. Different keys are generated for encryption (using **EnvelopedData**, Section 6.7, Enveloped Data) and authentication (using **AuthenticatedData**, Section 6.3, Authenticated Data).

7.2 Asymmetric Key Transport

In asymmetric key transport, the **RecipientInfo** contains an identifier of the recipient's public key certificate. This allows the retrieval of the associated private key, and the decryption of the CEK. ANS X9.63 [8] contains ANSI X9 approved key transport mechanisms (See also X9.44 (draft) [21]).

```
KeyTransRecipientInfo ::= SEQUENCE {
   version                  Version,
   rid                      RecipientIdentifier,
   keyEncryptionAlgorithm   KeyEncryptionAlgorithmIdentifier,
   encryptedKey             EncryptedKey
}

KeyEncryptionAlgorithmIdentifier ::=
         AlgorithmIdentifier {{ KeyEncryptionAlgorithms }}

RecipientIdentifier ::= CHOICE {
   issuerAndSerialNumber  IssuerAndSerialNumber,
   rKeyId                 [0] RecipientKeyIdentifier,
   certHash               [73] EXPLICIT Hash
}

KeyEncryptionAlgorithms ALGORITHM ::= {
   { OID rsaEncryption PARMS NullParms },
```



```
   ...  -- expect other key encryption algorithms --
}

EncryptedKey ::= OCTET STRING
```

The value of **version** is the schema version number. This value shall be ninety-six for this standard.

7.3 Asymmetric Key Agreement

In key agreement, the sender will derive a symmetric key encryption key (KEK) for each recipient, using the private key(s) of the sender and the public key of the recipient. The KEK is then used to encrypt the content-encryption key. Relevant parameters include the KEK algorithm identifier (unless it can be inferred from the key transport mechanism identifier), an additional ephemeral public key for the sender (for some ANS X9.42 [5] and ANS X9.63 variants), and a nonce. These may be carried in the **ukm** component of **KeyAgreeRecipientInfo**.

```
KeyAgreeRecipientInfo ::= SEQUENCE {
   version                  Version,
   originator               [0] EXPLICIT OriginatorIdentifierOrKey,
   ukm                      [1] EXPLICIT UserKeyingMaterial  OPTIONAL,
   keyEncryptionAlgorithm   KeyEncryptionAlgorithmIdentifier,
   recipientEncryptedKeys   RecipientEncryptedKeys
}

RecipientEncryptedKeys ::= SEQUENCE OF RecipientEncryptedKey

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

OriginatorIdentifierOrKey ::= CHOICE {
   issuerAndSerialNumber  IssuerAndSerialNumber,
   subjectKeyIdentifier   [0]  SubjectKeyIdentifier,
   originatorKey          [1]  OriginatorPublicKey,
   certHash               [73] EXPLICIT Hash
}

OriginatorPublicKey ::= SEQUENCE {
   algorithm  AlgorithmIdentifier {{ ... }},
   publicKey  BIT STRING
}
```

The value of **version** is the schema version number. This value shall be ninety-six for this standard.

The following ANS X9.42 and ANS X9.63 variants are particularly appropriate for use in this standard:

a) The sender generates an ephemeral key pair and sends the ephemeral public key to the recipient (in the **ukm** component). This key pair must use the same domain parameters as the recipient's certified key pair. The content-encryption key is then derived by the sender using the ephemeral key pair and the recipient's certified static key pair. The recipient should perform a validation of the originator's ephemeral public key, as described in ANS X9.42 and ANS X9.63. This scheme corresponds to the **dhOneFlow** scheme of ANS X9.42 and the 1-Pass Diffie-Hellman scheme of ANS X9.63. This variant does not provide data origin authentication and, therefore, should be used with signed-data and not authenticated-data. This variant provides forward secrecy.

b) A key may be derived using certified key pairs for the sender and recipient. This method provides data origin authentication and can be used with authenticated-data. The key is static for the life of the certificates, in this case, so exposure of either (sender or recipient) private key will reveal all messages between the sender and recipient, i.e. this method does not provide forward secrecy. This is an instance of the **dhStatic** model in ANS X9.42 and the static Unified Model scheme of ANS X9.63.

c) The mechanism of (b) can be used, but with the addition of a nonce (conveyed in the **`ukm`** component) which is used in deriving the KEK. This generates a different KEK for each message, but forward secrecy is not provided. This variant uses the **dhStatic** model of ANS X9.42, and the static Unified Model scheme of ANS X9.63 and can be used with authenticated-data.

7.4 Pre-established Key Encryption Keys

In pre-established key encrypting keys, the sender encrypts the content-encryption key under a shared KEK established by other means. No domain parameters are required for this mechanism.

```
KEKRecipientInfo ::= SEQUENCE {
   version                 Version,
   kekid                   KEKIdentifier,
   keyEncryptionAlgorithm  KeyEncryptionAlgorithmIdentifier,
   encryptedKey            EncryptedKey
}

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

The value of **`version`** is the schema version number. This value shall be ninety-six for this standard.

7.5 External Mechanisms – Constructive Key Management

The Constructive Key Management technique (CKM), described in ANS X9.69, is used to encrypt objects. It may be used with CMS to encrypt a message (as the object) to a set of users sharing a common set of values (known as key components). Access to the message content may be controlled by distributing subsets of these key components to users. The key components used for the encryption of a specific message are chosen by the sender, and these components define the intended recipients of the message. The sender-chosen components are combined with a random component to produce an object key to be used as the content-encryption key. CKM is particularly useful where the data flows among groups of users are well known and predefined.

8 Conformance Classes

The conformance classes for this standard include all of those defined in ANS X9.73. These classes are intended to simplify the procurement of conforming products, allowing implementations to state the classes to which they conform, and the algorithms that they support. One additional class is defined in this standard. That is support for the ASN.1 XML Encoding Rules (XER).

Annex A
(normative)

XML CMS Object Identifiers

This annex includes object identifiers for content types, attributes, and other objects used in this standard. Many of these are defined in other documents, but are included here for completeness.

```
XCMSObjectIdentifers {
   iso(1) identified-organization(3) tc68(133) country(16) x9(840)
      x9Standards(9) x9-96(96) module(0) oids(1) }
 DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS All; --

IMPORTS

   ALGORITHM
      FROM XMLCryptographicMessageSyntax {
         iso(1) identified-organization(3) tc68(133) country(16) x9(840)
            x9Standards(9) x9-96(96) module(0) xcms(2) } ;


OID ::= OBJECT IDENTIFIER  -- Alias

-- Content types, from PKCS #7 and 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) }

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 }

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

```
      attribute(1) namedkeyencryptedData(2)
}

-- Signed attributes, from PKCS #9, S/MIME, and ANS X9.73 --

id-contentType OID ::= { pkcs9 contentType(3) }

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

id-signingTime OID ::= { pkcs9 signingTime(5) }

id-contentIdentifier OID ::= {
   smime id-aa(2) contentIdentifier(7)
}

id-msgSequenceNo OID ::= {
   iso(1) member-body(2) us(840) x973(10060)
      attribute(1) msgSequenceNo(1)
}

id-signingCertificate OID ::= {
   smime  id-aa(2) signingCertificate(12)
}

id-otherSigningCert OID ::= {
   itu-t(0) identified-organization(4) etsi(0)
      electronic-signature-standard(1733) part1(1) attributes(1) 12
}

id-biometricSyntax OID ::= {
   iso(1) member-body(2) us(840) x973(10060)
      attribute(1) biometricSyntax(2)
}

-- Message component authenticated attribute --

id-messageComponents OID ::= {
   iso(1) identified-organization(3) tc68(133) country(16) x9(840)
      x9Standards(9) x9-96(96) attributes(1) messageComponents(1)
}

-- Authenticated attribute, from S/MIME --

id-macValue OID ::= {
   smime  aa(2) macValue(8)
}

-- Unsigned attribute, from PKCS #9 --

id-countersignature OID ::= {
    pkcs9 counterSignature(6) }

-- CKM key management object identifiers --
```

```
id-ckm-recip-info OID ::= {
   iso member-body(2) us(840) x973(10060) km(2) 1
}

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-other-cert-types OID ::= {
   iso member-body(2) us(840) x973(10060) other-cert-types(4)
}

id-x968-cert-type OID ::= {
   id-other-cert-types x968-domain-cert(1)
}

-- ANS X9.45 object identifiers --

id-signaturePurpose OID ::= {
   iso(1) member-body(2) us(840) x945(10052) signaturePurpose(23) }

-- FIPS 180-1 and FIPS 180-2 Secure Hash Algorithm --

sha-1 OID ::= {
   iso(1) identified-organization(3) oiw(14) secsig(3)
      algorithm(2) 26
}

sha2Algorithm OID ::= {
   joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101)
      csor(3) nistAlgorithm(4) hashAlgs(2)
}

id-sha256 OID ::= { sha2Algorithm sha256(1) }

id-sha384 OID ::= { sha2Algorithm sha384(2) }
```

```
id-sha512 OID ::= { sha2Algorithm sha512(3) }

SHA-Algorithms ALGORITHM ::= {

   -- The parameters associated with id-sha1, id-sha256, id-sha384,  --
   -- and id-sha512 should be omitted, but if present, shall have    --
   -- a value of ASN.1 type NULL. This is to align with the original --
   -- NIST definitions. For these SHA algorithms, implementations    --
   -- shall accept AlgorithmIdentifier values with NULL parameters   --
   -- and with the optional parameters component not present.        --

   { OID sha-1      PARMS NullParms }  |
   { OID id-sha256  PARMS NullParms }  |
   { OID id-sha384  PARMS NullParms }  |
   { OID id-sha512  PARMS NullParms },

   ...  -- Expect additional algorithms --
}

NullParms ::= NULL

-- X9.57 DSA signature generated with SHA-1 hash (DSA X9.30) --

dsa-with-sha1 OID ::=  {
   iso(1) member-body(2) us(840) x9-57(10040) x9algorithm(4) 3 }

--  X9.71 HMAC with SHA-1 algorithm --

hmac-with-SHA1 OID ::= {
   iso(1)  identified-organization(3) dod(6)
       internet(1) security(5) mechanisms(5) 8 1 2 }

-- RSA PKCS #1 signature generated with SHA-1 hash & encryption scheme --

sha1WithRSAEncryption OID  ::=  {
   iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) 1 5 }

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

-- ANS X9.52 Triple DES Modes of Operation --

des-ede3-cbc OID ::= {
   iso(1) member-body(2) us(840) rsadsi(113549)
      encryptionAlgorithm(3) 7
}

-- X9.62 ECDSA signature with SHA-1 --

ecdsa-with-SHA1 OID ::= {
   iso(1) member-body(2) us(840) ansi-x962(10045) signatures(4) 1 }


END -- XCMSObjectIdentifiers --
```

Annex B
(normative)

XML CMS Schema

This annex contains an ASN.1 schema for the XML markup representation of the ANS X9.96 XML Cryptographic Message Syntax. This schema is based on that defined in the ANS X9.73 CMS standard and can be used to produce identical binary encoded messages as that standard. While ANS X9.73 defines a compact binary encoding of CMS messages using the Basic Encoding Rules (BER) of ASN.1, this Standard specifies an XML markup encoding of these same abstract values using the XML Encoding Rules (XER) of ASN.1.

```
XMLCryptographicMessageSyntax {
   iso(1) identified-organization(3) tc68(133) country(16) x9(840)
      x9Standards(9) x9-96(96) module(0) xcms(2) }
   DEFINITIONS IMPLICIT TAGS ::= BEGIN

-- EXPORTS All; --

IMPORTS

   -- ITU-T Rec. X.509 | ISO/IEC 9594-8 CertificateExtensions --

   PolicyInformation
      FROM CertificateExtensions {
         joint-iso-itu-t ds(5) module(1) certificateExtensions(26) 4 }

   -- ANS X9.84-2003 X9-84-Biometrics --

   BiometricSyntax, NamedKeyEncryptedData
      FROM X9-84-Biometrics {
         iso(1) identified-organization(3) tc68(133)
            country(16) x9(840) x9Standards(9)
               x9-84(84) module(0) biometrics(1) rev(1) }

   -- ANS X9.45 X945-EnhancedManagement --

   SignaturePurposes
      FROM X945-EnhancedManagement {
         iso(1) member-body(2) us(840) x945(10052) modules(0)
            enhanced-management(0) }

   ExtendedKeyMgmtRecipientInfo, IssuerAndSerialNumber,
   SigningCertificate
      FROM X973CryptographicMessageSyntax {
         iso(1) member-body(2) us(840) x973(10060) module(0) 1 }

   -- ANS X9.96 XCMSObjectIdentifiers --

   des-ede3-cbc, dsa-with-sha1, ecdsa-with-SHA1, hmac-with-SHA1,
   id-biometricSyntax, id-contentIdentifier, id-contentType,
   id-countersignature,id-ct-authData, id-data, id-digestedData,
```

```
   id-encryptedData, id-envelopedData, id-macValue,
   id-messageDigest, id-msgSequenceNo, id-namedkeyencryptedData,
   id-otherSigningCert, id-signaturePurpose, id-signedData, id-signingCertificate,
   id-signingTime, id-x968-cert-type, NullParms, OID, rsaEncryption, SHA-Algorithms,
   sha1WithRSAEncryption, sha-1

      FROM XCMSObjectIdentifers {
         iso(1) identified-organization(3) tc68(133) country(16)
            x9(840) x9Standards(9) x9-96(96) module(0) oids(1) };


EncapsulatedContentInfo ::= SEQUENCE {
   eContentType  ContentType,
   eContent      [0] EXPLICIT
                    CONTENTS.&Type({Contents}{@eContentType})  OPTIONAL
}

ContentType ::= CONTENTS.&id({Contents})

CONTENTS ::= TYPE-IDENTIFIER  -- Defined in ISO/IEC 8824-2, Annex A

Contents CONTENTS ::= {
   { ESignedData             IDENTIFIED BY  id-signedData            } |
   { EEnvelopedData          IDENTIFIED BY  id-envelopedData         } |
   { EAuthenticatedData      IDENTIFIED BY  id-ct-authData           } |
   { EDigestedData           IDENTIFIED BY  id-digestedData          } |
   { EEncryptedData          IDENTIFIED BY  id-encryptedData         } |
   { ENamedKeyEncryptedData  IDENTIFIED BY  id-namedkeyencryptedData } |
   { EData                   IDENTIFIED BY  id-data                  },

   ...  -- Expect additional objects --
}

ESignedData ::= OCTET STRING (CONTAINING SignedData)

EEnvelopedData ::= OCTET STRING (CONTAINING EnvelopedData)

EAuthenticatedData ::= OCTET STRING (CONTAINING AuthenticatedData)

EDigestedData ::= OCTET STRING (CONTAINING DigestedData)

EEncryptedData ::= OCTET STRING (CONTAINING EncryptedData)

ENamedKeyEncryptedData ::= OCTET STRING (CONTAINING NamedKeyEncryptedData)

EData ::= OCTET STRING (CONTAINING Data)

Data ::= OCTET STRING

SignedData ::= SEQUENCE {
   version           Version,
   digestAlgorithms  DigestAlgorithmIdentifiers,
   encapContentInfo  EncapsulatedContentInfo,
   certificates      [0] CertificateSet  OPTIONAL,
```

```
   crls              [1] CertificateRevocationLists  OPTIONAL,
   signerInfos       SignerInfos
}

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

DigestAlgorithms ALGORITHM ::= {
   SHA-Algorithms,

   ... -- Expect other digest algorithms  --
}

SignerInfos ::= SET OF SignerInfo

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

SignerIdentifier ::= CHOICE {
   issuerAndSerialNumber  IssuerAndSerialNumber,
   subjectKeyIdentifier   [0] SubjectKeyIdentifier,
   certHash               [73] EXPLICIT Hash
}

DigestAlgorithmIdentifier ::= AlgorithmIdentifier {{DigestAlgorithms}}

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier {{SignatureAlgorithms}}

SignatureAlgorithms ALGORITHM ::= {
   { OID dsa-with-sha1          PARMS NullParms } |
   { OID ecdsa-with-SHA1        PARMS NullParms } |
   { OID sha1WithRSAEncryption  PARMS NullParms },

   ...  -- Expect other signature algorithms --
}

SignedAttributes ::= SET SIZE(1..MAX) OF SignedAttribute

SignedAttribute ::= Attribute {{Signed}}

Signed ATTRIBUTE ::={
   { WITH SYNTAX ContentType              ID id-contentType        }  |
   { WITH SYNTAX MessageDigest            ID id-messageDigest      }  |
   { WITH SYNTAX SignaturePurposes        ID id-signaturePurpose   }  |
   { WITH SYNTAX SigningTime              ID id-signingTime        }  |
   { WITH SYNTAX SigningCertificate       ID id-signingCertificate }  |
   { WITH SYNTAX OtherSigningCertificate  ID id-otherSigningCert   }  |
   { WITH SYNTAX BiometricSyntax          ID id-biometricSyntax    }  |
```

```
   { WITH SYNTAX MsgSequenceNo            ID id-msgSequenceNo      }  |
   { WITH SYNTAX Content                  ID id-contentIdentifier  }  |
   { WITH SYNTAX MessageComponents        ID id-messageComponents  },

   ...  -- Expect additional objects --
}

MessageDigest ::= OCTET STRING

SigningTime ::= CHOICE {
   utcTime          UTCTime,
   generalizedTime  GeneralizedTime
}

OtherSigningCertificate ::= SEQUENCE {
   certs     OtherCertIDs,
   policies  PolicyInfos  OPTIONAL
}

OtherCertIDs ::= SEQUENCE OF OtherCertID

OtherCertID ::=  SEQUENCE {
   certHash      Hash,
   issuerSerial  IssuerAndSerialNumber  OPTIONAL
}

Hash ::= CHOICE {
   ietf       CertHash,  -- SHA-1 hash of entire certificate
   withAlgID  DigestInfo
}

CertHash ::= OCTET STRING (ENCODED BY sha-1)

PolicyInfos ::= SEQUENCE OF PolicyInformation

DigestInfo ::= SEQUENCE {
   hashAlgorithm  DigestAlgorithmIdentifier,
   digest         OCTET STRING
}

MsgSequenceNo ::= INTEGER (0..MAX)

Content ::= OCTET STRING

MessageComponents ::= SEQUENCE SIZE(1..MAX) OF Component

Component ::= UTF8String
UnsignedAttributes ::= SET SIZE(1..MAX) OF UnsignedAttribute

UnsignedAttribute ::= Attribute {{Unsigned}}

Countersignature ::= SignerInfo

Unsigned ATTRIBUTE ::= {
```

```
   { WITH SYNTAX Countersignature ID id-countersignature }  |
   { WITH SYNTAX BiometricSyntax  ID id-biometricSyntax  },

   ...  -- Expect additional objects --
}

Attribute { ATTRIBUTE:IOSet } ::= SEQUENCE {
   type    ATTRIBUTE.&id({IOSet}),
   values  SET OF ATTRIBUTE.&Type({IOSet}{@type})
}

SignatureValue ::= OCTET STRING

EnvelopedData ::= SEQUENCE {
   version               Version,
   originatorInfo        [0] OriginatorInfo  OPTIONAL,
   recipientInfos        RecipientInfos,
   encryptedContentInfo  EncryptedContentInfo,
   unprotectedAttrs      [1] UnprotectedAttributes  OPTIONAL
}

UnprotectedAttributes ::= SET SIZE(1..MAX) OF Attribute {{Unprotected}}

Unprotected ATTRIBUTE ::= { ...  -- Expect additional objects -- }

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

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

EncryptedContentInfo ::= SEQUENCE {
   contentType                 ContentType,
   contentEncryptionAlgorithm  ContentEncryptionAlgorithmIdentifier,
   encryptedContent            [0] EncryptedContent  OPTIONAL
}

ContentEncryptionAlgorithmIdentifier ::=
                     AlgorithmIdentifier {{ContentEncryptionAlgorithms}}

ContentEncryptionAlgorithms ALGORITHM ::= {
   { OID des-ede3-cbc PARMS IV },

   ...  -- Expect other content encryption algorithms --
}

IV ::= OCTET STRING (SIZE(8))

EncryptedContent ::= OCTET STRING

RecipientInfo ::= CHOICE {
   ktri   KeyTransRecipientInfo,
   kari   [1] KeyAgreeRecipientInfo,
```

```
   kekri  [2] KEKRecipientInfo,
-- Choice [3] reserved for IETF password-based encryption
   ori    [4] ExtendedKeyMgmtRecipientInfo
}

EncryptedKey ::= OCTET STRING

KeyTransRecipientInfo ::= SEQUENCE {
   version                 Version,
   rid                     RecipientIdentifier,
   keyEncryptionAlgorithm  KeyEncryptionAlgorithmIdentifier,
   encryptedKey            EncryptedKey
}

KeyConstructionAlgorithmIdentifier ::=
         AlgorithmIdentifier {{KeyConstructionAlgorithms}}

KeyConstructionAlgorithms ALGORITHM ::= { ...  -- Expect additional objects -- }

KeyEncryptionAlgorithmIdentifier ::=
         AlgorithmIdentifier {{KeyEncryptionAlgorithms}}

KeyEncryptionAlgorithms ALGORITHM ::= {
   { OID rsaEncryption PARMS NullParms },

   ...  -- expect other key encryption algorithms --
}


OriginatorIdentifierOrKey ::= CHOICE {
   issuerAndSerialNumber  IssuerAndSerialNumber,
   subjectKeyIdentifier   [0]  SubjectKeyIdentifier,
   originatorKey          [1]  OriginatorPublicKey,
   certHash               [73] EXPLICIT Hash
}

OriginatorPublicKey ::= SEQUENCE {
   algorithm  AlgorithmIdentifier {{ ... }},
   publicKey  BIT STRING
}

KeyAgreeRecipientInfo ::= SEQUENCE {
    version                 Version,
    originatorCert          [0] EXPLICIT OriginatorIdentifierOrKey,
    ukm                     [1] EXPLICIT UserKeyingMaterial  OPTIONAL,
    keyEncryptionAlgorithm  KeyEncryptionAlgorithmIdentifier,
    recipientEncryptedKeys  RecipientEncryptedKeys
}

RecipientEncryptedKeys ::= SEQUENCE OF RecipientEncryptedKey

RecipientEncryptedKey ::= SEQUENCE {
   rid           RecipientIdentifier,
   encryptedKey  EncryptedKey
```

```
}

RecipientIdentifier ::= CHOICE {
   issuerAndSerialNumber  IssuerAndSerialNumber,
   rKeyId                 [0] RecipientKeyIdentifier,
   certHash               [73] EXPLICIT Hash
}

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

SubjectKeyIdentifier ::= OCTET STRING

KEKRecipientInfo ::= SEQUENCE {
   version                 Version,
   kekid                   KEKIdentifier,
   keyEncryptionAlgorithm  KeyEncryptionAlgorithmIdentifier,
   encryptedKey            EncryptedKey
}

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

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

Digest ::= OCTET STRING

EncryptedData ::= SEQUENCE {
   version               Version,
   encryptedContentInfo  EncryptedContentInfo
}

AuthenticatedData ::= SEQUENCE {
   version                      Version,
   originatorInfo               [0] OriginatorInfo  OPTIONAL,
   recipientInfos               RecipientInfos,
   macAlgorithm
MACAlgorithmIdentifier,
   digestAlgorithm              [1] DigestAlgorithmIdentifier  OPTIONAL,
   encapContentInfo             EncapsulatedContentInfo,
   authenticatedAttributes      [2] AuthAttributes  OPTIONAL,
   mac                          MessageAuthenticationCode,
   unauthenticatedAttributes    [3] UnauthAttributes  OPTIONAL
```

```
}

MACAlgorithmIdentifier ::= AlgorithmIdentifier {{MACAlgorithms}}

MACAlgorithms ALGORITHM ::= {
   { OID hmac-with-SHA1 },

   ...  -- expect other MAC or HMAC algorithms --
}

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

Authenticated ::= Signed

MACValue ::= OCTET STRING

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

Unauthenticated ::= Unsigned

MessageAuthenticationCode ::= OCTET STRING

CertificateRevocationLists ::= OCTET STRING

CertificateSet ::= OCTET STRING

Version ::= INTEGER { vx9-96(96) } ( vx9-96, ... )

UserKeyingMaterials ::= SET SIZE(1..MAX) OF UserKeyingMaterial

UserKeyingMaterial ::= OCTET STRING

OtherKeyAttribute ::= AttributeTypeAndValue

AttributeTypeAndValue ::= SEQUENCE {
   type   ATTRIBUTE.&id({OtherAttributes}),
   value  ATTRIBUTE.&Type({OtherAttributes}{@type})
}

OtherAttributes ATTRIBUTE ::= { ...  -- Expect additional objects -- }

-- Supporting definitions --

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

ALGORITHM ::= CLASS {
   &id    OBJECT IDENTIFIER  UNIQUE,
   &Type  OPTIONAL
}
```

```
  WITH SYNTAX { OID &id [ PARMS &Type ] }

AlgorithmIdentifier { ALGORITHM:IOSet } ::= SEQUENCE {
   algorithm   ALGORITHM.&id({IOSet}),
   parameters  ALGORITHM.&Type({IOSet}{@algorithm})  OPTIONAL
}

END -- XMLCryptographicMessageSyntax --
```

Bibliography

[21] ANS X9.44 (draft), *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Factoring-based Cryptography*

[22] ISO 15782-1:2003, Banking - Certificate Management Part 1: Public Key Certificates.

[23] ISO 15782-2:2002, Banking - Certificate Management Part 1: Certificate Extensions.

[24] ISO/IEC 9594-8: Information technology | ITU-T Recommendation X.509, Open Systems Interconnection -- The Directory: Authentication framework", International Organization for Standardization, Geneva, Switzerland, 2000.