Designation: E 2084 – 00

An American National Standard

Standard Specification for Authentication of Healthcare Information Using Digital Signatures[1]

This standard is issued under the fixed designation E 2084; the number immediately following the designation indicates the year of original adoption or, in the case of revision, the year of last revision. A number in parentheses indicates the year of last reapproval. A superscript epsilon (ε) indicates an editorial change since the last revision or reapproval.

1. Scope

1.1 This specification covers the use of digital signatures to provide authentication of healthcare information, as described in Guide E 1762. It describes how the components of a digital signature system meet the requirements specified in Guide E 1762. This includes specification of allowable signature and hash algorithms, management of public and private keys, and specific formats for keys, certificates, and signed healthcare documents.

1.2 This specification should be read in conjunction with Guide E 1762, which describes the scope of, and requirements for, authentication of healthcare information. This specification describes one implementation (digital signatures) that meets all of the requirements of Guide E 1762. It does not prescribe any particular policy regarding which documents shall be authenticated, and by whom.

2. Referenced Documents

2.1 *ASTM Standards:*

E 1762 Guide for Electronic Authentication of Healthcare Information[2]

2.2 *ANSI Standards:*[3]

X9.30 Part 2: Public Key Cryptography Using Irreversible Algorithms: Secure Hash Algorithm (SHA-1)

X9.31 Reversible Digital Signature Algorithms

X9.55 Extensions to Public Key Certificates and CRLs

X9.57 Certificate Management

X9.62 Elliptic Curve Digital Signature Algorithm

2.3 *ISO Standards:*[4]

ISO 9594–8 1993: The Directory: Authentication Framework (also available as ITU-S X.509)

ISO 8824–1 1993: Specification of Abstract Syntax Notation One (ASN.1)

ISO 8825–1 1993: Specification of Basic Encoding Rules for ASN.1

ISO 9796 1991: Digital Signature Scheme Giving Message Recovery

ISO 10166 1991: Document Filing and Retrieval (DFR)

2.4 *Internet Standards:*[5]

RFC 2630 Cryptographic Message Syntax

2.5 *Other Documents:*[6]

RSA Laboratories, "PKCS #1: RSA Encryption Standard (version 1.5)," November 1993

RSA Laboratories, "PKCS #5: Password Based Encryption (version 1.5)," November 1993

RSA Laboratories, PKCS #6: Extended Certificate Syntax Notation

RSA Laboratories, "PKCS #7: Cryptographic Message Syntax (version 1.5)," November 1993

RSA Laboratories, PKCS #9: Selected Attribute Types

ITU-T X.501 Information Technology Open Systems Interconnection—The Directory: Models

3. Terminology

3.1 *Definitions:*

3.1.1 *attribute*—piece of information associated with the use of a document.

3.1.2 *authentication (data origin)*—corroboration that this source of data received is as claimed.

3.1.3 *authentication (user)*—provision of assurance of the claimed identity of an entity.

3.1.4 *certificate (public key)*—digitally signed data structure that binds a user's identity to a public key.

3.1.5 *data integrity*—property that data has not been altered or destroyed in an unauthorized manner.

3.1.6 *digest*—result of applying a one-way hash function to a message.

3.1.7 *digital signature*—data associated with, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery, for example, by the recipient.

---

[1] This specification is under the jurisdiction of ASTM Committee E31 on Healthcare Informatics and is the direct responsibility of Subcommittee E31.20 on Data and System Security for Health Information.

Current edition approved April 10, 2000. Published June 2000.

[2] *Annual Book of ASTM Standards*, Vol 14.01.

[3] Available from American National Standards Institute, 11 W. 42 nd St., 13th Floor, New York, NY 10036.

[4] Available from ISO, 1 Rue de Varembe, Case Postale 56, CH 1211, Geneve, Switzerland.

[5] Available at http://www.ietf.org.

[6] Available from RSA Data Security, 100 Marine Parkway, Redwood City, CA 94605.

3.1.8 *document access time*—time(s) when the subject document was accessed for reading, writing, or editing.

3.1.9 *document attribute*—attribute describing a characteristic of a document. **E 1762**

3.1.10 *document creation time*—time of the creation of the subject document **E 1762**

3.1.11 *document editing time*—time(s) of the editing of the subject document. **E 1762**

3.1.12 *electronic document*—defined set of digital information, the minimal unit of information which may be digitally signed. **E 1762**

3.1.13 *event time*—the time of the documented event.

3.1.14 *(one-way) hash function*—function which maps strings of bits to fixed-length strings of bits, satisfying the following two properties: *(1)* it is computationally infeasible to find for a given output an input which maps to this output; *(2)* it is computationally infeasible to find for a given input a second input which maps to the same output.

3.1.15 *private key*—key in an asymmetric algorithm; the possession of this key is restricted, usually to one entity.

3.1.16 *public key*—key in an asymmetric algorithm that is publicly available.

3.1.17 *repudiation*—denial by one of the entities involved in a communication of having participated in all or part of the communication.

3.1.18 *role*—role of a user when performing a signature. Examples include: physician, nurse, allied health professional, transcriptionist/recorder, and others. **E 1762**

3.1.19 *signature attribute*—attribute characterizing a given user's signature on a document **E 1762**

3.1.20 *signature purpose*—indication of the reason an entity signs a document. This is included in the signed information and can be used when determining accountability for various actions concerning the document. Examples include: author, transcriptionist/recorder, and witness. **E 1762**

3.1.21 *signature time*—time a particular signature was generated and affixed to a document. **E 1762**

3.1.22 *signature verification*—process by which the recipient of a document determines that the document has not been altered and that the signature was affixed by the claimed signer. This will in general make use of the document, the signature, and other information such as cryptographic keys or biometric templates.

3.2 *Acronyms: Acronyms:*

3.2.1 *ABA*—American Bar Association

3.2.2 *ASC X9*—Accredited Standards Committee X9

3.2.3 *CA*—Certificate Authority

3.2.4 *CEN*—Comité Européan de Normalisation (European Standards Committee)

3.2.5 *CRL*—Certificate Revocation List

3.2.6 *DSA*—Digital Signature Algorithm

3.2.7 *ISO*—International Standards Organization

3.2.8 *NIST*—National Institute for Standards and Technology

3.2.9 *RSA*—Rivest, Shamir, and Adleman

4. Technology Overview

4.1 Digital signatures are a cryptographic technique in which each user is associated with a pair of keys. The private key is kept secret, while the public key is distributed to the potential verifiers of the user's digital signature. To sign a document, the document and private key are input to a cryptographic process which outputs a bit string (the signature). To verify a signature, the signature, the document, and the user's public key are input to a cryptographic process, which returns an indication of success or failure. Any modification to the document after it is signed will cause the signature verification to fail (integrity). If the signature was computed using a private key other than the one corresponding to the public key used for verification, the verification will fail (authentication).

4.2 A digital signature is thus a piece of data associated with a document which allows the recipient to prove the origin of the document and to protect against forgery. Digital signatures are formed using asymmetric encryption algorithms as described in 4.1. Digital signatures are associated with a document.

4.3 To sign a message, it is first hashed into a single block using a one-way hash function. A one-way hash function has the property that, given the digest (the output of the hash function), it is computationally infeasible to construct any message that hashes to that value, or to find two messages that hash to the same digest. The digest is processed with the user's private key, and the result may be appended to the message as its signature.

4.4 Separating the signature from the message reduces the amount of data to be encrypted to a single block. This is important since public key algorithms are generally substantially slower than symmetric algorithms. The signature process also introduces redundancy into the message. Redundancy allows the recipient to detect unauthorized changes to the message. Most messages already contain sufficient redundancy to detect such a forgery (such as, English text, time stamps, etc.). The signature process adds additional redundancy, since the message must also hash to the specified digest.

4.5 To verify a signature, the received message is digested, and the digest is processed along with the public key and signature. The result is an indication of success or failure of the verification.

4.6 A digital signature provides the following security services:

4.6.1 Integrity, since any modification of the data being signed will result in a different digest, and thus a different signature.

4.6.2 Origin authentication, since only the holder of the private key corresponding to the public key used for validation could have signed the message.

4.6.3 Support for non-repudiation, that is, irrevocable proof to a third party that only the signer could have created the signature.

4.7 A canonical representation for documents shall be specified for use by cryptographic mechanisms described in this specification. In general, a document may be stored on a system in a different form than the one in which it was generated. For example, if the document contains many numeric values, some systems may store them as integers but

others as text. Since signatures are computed over representations (encodings), rather than abstract values, there shall be a specific representation the signature is computed over. Such a representation can be defined using Abstract Syntax Notation One (ASN.1) (ISO 8824–1) with an appropriate set of encoding rules, such as the Distinguished Encoding Rules specified in ISO 8825–1. As an added benefit, a number of existing ISO and ANSI standards (notably X.509 and ANSI X9.57) are available that define ASN.1 structures for digital signatures. These are presented in Section 9. The sender and recipient shall agree on an encoding mechanism for the (signed) document. Other mutually agreed or standard or standard encoding mechanisms may be used. If ASN.1 encoding is used, then the specification in Section 9 shall be used. This document representation is taken from ISO 10166.

4.8 A document may be signed by one or more users. Each user's signature and other information is contained in a separate structure. Each signature structure contains an indication of the public key needed to validate the signature and a bit string containing the actual signature. Additionally, other information relevant to the particular signer would be included in an individual signature computation. This per-signer information would be included in the signature computation as *signature attributes*. A signature structure may also include per-signer information which is not signed, but merely appended to the signature structure (*unsigned attributes*). An important unsigned attribute is the *countersignature*. A countersignature is a signature on the signature structure in which it is found, rather than on the document itself. A countersignature thus provides proof of the order in which signatures were applied. Since the countersignature is itself a signature structure, it may itself contain countersignatures; this allows construction of arbitrarily long chains of countersignatures.

5. Mapping to Requirements

5.1 This section maps the algorithms, procedures, and data formats described in Sections 6-9 to the electronic signature requirements presented in Guide E 1762.

5.2 *General Requirements*:

5.2.1 *Non-repudiation*—This is provided by a digital signature, assuming the private key used for signing is never divulged (see Section 7). Additionally, reliable binding of keys to names shall be provided, using certificates. Appropriate secure archive facilities shall be provided for expired and revoked keys and certificates, as well as CRLs. This might be done by using the new private key to sign the old public key in a certificate.

5.2.2 *Integrity*—This is provided by a digital signature, since any modification of the signed document or signature causes signature verification to fail.

5.3 *User Authentication Requirements*:

5.3.1 *Secure User Authentication*—This shall be provided by the system implementing digital signature. Secure user authentication is the topic of several standards currently under development by ASTM Committee E31.

5.4 *Logical Manifestation Requirements*:

5.4.1 *Multiple Signatures*—These are provided, on a document by the **SignerInfo** structures described in Section 9. Additionally, a given signature may have multiple countersignatures.

5.4.2 *Signature Attributes*—These are provided in the **SignerInfo** structure as the **authenticatedAttributes** component.

5.4.3 *Countersignatures*—These are provided using the **countersignature** unsigned attribute.

5.5 *Verification Requirements*:

5.5.1 *Transportability*—The ASN.1 specification in Section 9 defines a unique encoding for the signed document, which can be transported to another system for verification.

5.5.2 *Interoperability*—Any recipient which implements the ASN.1 specification of Section 9 and the appropriate signature algorithms can verify a signed document.

5.5.3 *Independent Verifiability*—A digital signature is verified using the public key of the signer, which can be certified by a trusted third party (CA) and distributed via a variety of mechanisms.

5.5.4 *Continuity of Signature Capability*—Publication of the public key used for signature verification does not allow an adversary to determine the private key used for signature.

6. Algorithms

6.1 While it is not the intent of this specification to restrict the use of cryptographically sound signature algorithms, it is desirable to limit the number of allowable algorithms, to ensure interoperability. This specification recommends use of the following algorithms:

6.1.1 RSA and variants, as described in ANSI X9.31, PKCS #1, andAppendix X1, or

6.1.2 DSA and variants, as described in ANSI X9.30, ANSI X9.62, andAppendix X2.

6.2 Allowable hash algorithms for use with the signature algorithms include:

6.2.1 For RSA and variants, SHA-1.

6.2.2 For DSA and variants, SHA-1.

7. Public Key Management

7.1 To verify a signature, a user shall obtain the signer's public key from a source that he or she trusts. This source is a public key certificate, which binds a user's name to his public key. Certificates are signed by a trusted issuer, the Certification Authority (CA). Besides the user's name and public key, the certificate contains the issuing CA's name, a serial number, and a validity period. The format of a certificate is specified in ISO 9594–8 and in ANSI X9.57. Section 9 of this specification describes this format in more detail.

7.2 Although this specification does not impose any particular structure on the CAs, many implementations find it reasonable to impose a hierarchical structure. A hierarchy of CAs can be set up, where the higher level CAs sign the certificates of the CAs beneath them, etc. The lowest level of CAs sign user certificates. At the top of this hierarchy are relatively few CAs (perhaps one per country) who may "cross-certify" each other's public keys.

7.3 Various security architectures define mechanisms to construct a certification path through the hierarchy to obtain a given user's certificate and all CA certificates necessary to

validate it. These architectures share the common characteristics that a user need only trust one other public key in order to obtain and validate any other certificate. The trusted key may be that of the top-level CA (in a centralized trust model), or the local CA that issued the user's certificate (in a decentralized model). Since there will be a variety of entities offering certification services (ranging from commercial entities to the U.S. government), implementations may find it useful to accommodate multiple trusted public keys, in the absence of cross-certification between all of these services.

7.4 Certificates contain an expiration date. If it is necessary to cancel a certificate prior to its expiration date (for example, if the name association becomes invalid or the corresponding private key is lost or compromised), the certificate may be added to the CA's certificate revocation list (CRL). This list is signed by the CA and widely distributed (for example, as part of the CA's directory entry). Each entry contains the revoked certificate's serial number, a revocation time, and optionally a revocation reason and time of suspected compromise. The certificate remains on the CRL until its expiration date. A system will typically archive expired certificates and CRLs in order to be able to verify signatures after the fact. Such archives shall be protected from modification, perhaps by using write-once media such as optical disk, or by digitally signing the archive files. In the latter case, the files may periodically need to be re-signed as the certificates needed to verify the signatures expire.

7.5 Certificates and CRLs for use with this specification are defined in ANSI X9.57 and ANSI X9.55. The relevant ASN.1 specification can be found in Section 9.

8. Private Key Management

8.1 To support a non-repudiation service, the user's private key shall be protected from disclosure to other users. This ensures that only that user could have created a digital signature using the private key; therefore the user cannot repudiate a signature by claiming that someone else applied it.

8.2 The most secure way to protect the private key is to embed it in a tamperproof cryptographic module that will perform the signature computation internally. Such modules might include smart cards, cryptographic diskettes, and PCMCIA cards. Access to the signature function requires the user to authenticate himself to the module using passwords, PINs, or biometric controls (even including graphic signature verification), or a combination thereof. This approach is recommended by this specification.

8.3 A less secure way to protect the private key is to encrypt it under a secret key computed from a password entered by the user. The minimum allowable key length for such a secret key is 56 bits (the key length for the Data Encryption Standard). Use of the password also authenticates the user to the signing system. (One such mechanism is described in PKCS #5.) The encrypted private key may be stored on removable media like a floppy disk and decrypted when needed to perform a signature. This approach is acceptable, but not recommended by this specification.

9. ASN.1 Specification

9.1 Following is the ASN.1 specification for the data structures defined in this specification. Complete specifications for ASN.1 can be found in ISO 8824–1 and 8825–1.

9.2 *Document Structure and Attributes*:

9.2.1 A document is represented using the following ASN.1 type. This representation is only required when computing or verifying a signature on the document, and places no constraints on how the document is represented during transmission or storage (although ASN.1 might also be appropriate for these purposes). Attributes may be specified, if desired, using the ATTRIBUTE class that follows, or that in ITU-T X.501.

```
Document::= SEQUENCE {
    attributes          SET OF Attribute,
    content             CHOICE {
            digest              DigestedDocument,
            full                OCTET STRING }}

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

ATTRIBUTE CLASS:: = {
    &Syntax,
    &singleValued               BOOLEAN DEFAULT FALSE,
    &id                         OBJECT IDENTIFIER UNIQUE
}
WITH SYNTAX {
    WITH ATTRIBUTE SYNTAX&Syntax

    ID                          &id
}
DigestedDocument::= SEQUENCE {
    algorithm           Algorithmidentifier,
}
DigestedDocument::= SEQUENCE {
    algorithm           Algorithmidentifier,
    digest              OCTET STRING,
    locator   DocumentLocator OPTIONAL}

DocumentLocator ::= CHOICE {
    uri                         IA5String,
    dfr                         DFRName, –using ″uri″ attribute
    oid                         OBJECT IDENTIFIER,
    other                       INSTANCE OF OTHER-LOCATOR }

DFRName ::= SEQUENCE {                  –OSI Document Filing/Retrieval ID
    docStore            AETitle,
    upi                 OCTET STRING }

AETitle ::= CHOICE {                            –OSI application name/address
    nameForm            Name, – last RDN is ″qualifier″
    oidForm             SEQUENCE {
            title       OBJECT IDENTIFIER,
            qualifier   INTEGER }}

OTHER-LOCATOR ::= TYPE-IDENTIFIER
```

9.2.2 This structure specifies the physical representation of a document when presented to the authorization mechanism. This does not imply that the document shall be stored, transmitted, or otherwise manipulated using this representation at any time other than authorization processing. However, it may be efficient to store the attributes and digest as an ASN.1 encoded structure.

9.2.3 The **digest** alternative contains the document attributes and the digest of the document, along with an optional document locator. The **full** alternative contains the actual document content, as an OCTET STRING.

9.2.4 A document locator is used to access the full content of a document. Alternatives include:

9.2.4.1 *Uniform Resource Identifier (URI)*, which can be used to access the document via Internet protocols such as HTTP, FTP, and gopher.

9.2.4.2 *Document Reference*, for the OSI Document Filing and Retrieval (DFR) protocol ISO 10166. A reference consists of the application name and address of the document store and a unique permanent identifier for the document.

9.2.4.3 *OBJECT IDENTIFIER*, the application shall have a prior knowledge of the address where the document can be found.

9.2.4.4 *Custom Identifiers*, defined by an application (TYPE-IDENTIFIERs).

9.2.5 All documents include at least the document type attribute, which indicates the type and semantics of the document.

9.2.6 The document attributes defined in Guide E 1762 are defined as follows:

```
document-type ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              DocumentType
    ID                                 id-transaction-type }

DocumentType ::= OBJECT IDENTIFIER
location ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              Location
    ID                                 id-location }

Location ::= CHOICE {
    PresentationAddress,
    IPAddress
    X121Address,
    FreeFormAddress }

PresentationAddress ::= SEQUENCE { —for OSI applications
    pSelector [0] OCTET STRING OPTIONAL,
    sSelector [1] OCTET STRING OPTIONAL,
    tSelector  [2] OCTET SPRING OPTIONAL,
    nAddress [3] SEQUENCE OF OCTET STRING }

IPAddress ::= OCTET STRING for Internet, etc.

X121Address ::= NumericString (SIZE (1..15)) –X2.5

FreeFormAddress ::= OCTET STRING

patient-id ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              GeneralName
    ID                                 id-patient-id }

event-id ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              OBJECT IDENTIFIER
    ID                                 id-event-id }

amendment-to ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              DigestedDocument
    ID                                 id-amendment-to }

data-type ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              OBJECT IDENTIFIER
    ID                                 id-data-type }

data-format ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              OBJECT IDENTIFIER
    ID                                 id-data-format }

originating-organization ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              GeneralName,
    ID                                 id-orig-org }

event-time ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              GeneralizedTime,
    ID                                 id-event-time }

doc-creation-time ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              GeneralizedTime,
    ID                                 id-creation-time }

doc-modification-time ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              GeneralizedTime,
    ID                                 id-modif-time }

doc-access-time ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              GeneralizedTime,
    ID                                 id-access-time }

doc-identifier ATTRIBUTE ::= {
    WITH ATTRIBUTE-SYNTAX              OCTET STRING,
    ID                                 id-doc-identifier }
```

9.3 *Digital Signatures and Certificates*:

9.3.1 This section defines parameterized types used to construct digital signatures on arbitrary data types. These include public key formats (which are encapsulated in BIT STRINGs for transmittal), algorithm parameters, and signature formats (also encapsulated in BIT STRINGS).

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

SupportedAlgorithms ALGORITHM ::= {dsa | dsa-with-sha-1 | rsa-signature
| rsa-signature-with–sha–1 | rsaEncryption | rsaEncryptionWithSHA1 | ec-
PublicKeyAlgorithm | ecdsa–with-SHA1}

ALGORITHM :: = TYPE-IDENTIFIER

DSAPublicKey :: = INTEGER

DSAParameters ::= SEQUENCE {
                                  – length of p in bits
    prime1      INTEGER, – modulus p
    prime2      INTEGER, – modulus q
    base        INTEGER } – base g

RSAPublicKey ::= SEQUENCE {
    modulus     INTEGER
    exponent    INTEGER }

HASHED (ToBeHashed} ::= OCTET STRING (CONSTRAINED BY {
–must be the result of applying a hashing procedure to the
–DER-encoded octets of a value of
    ToBeHashed }

ENCRYPTED {ToBeEnciphered} ::= BIT STRING (CONSTRAINED BY {
–must be the result of applying an enciphered procedure to the
–DER-encoded octets of a value of
    ToBeEnciphered }

SIGNED {ToBeSigned}::=SEQUENCE {
    date            ToBeSigned,
signature COMPONENTS OF SIGNATURE {ToBeSigned} }

SIGNATURE {OfSignature} ::=SEQUENCE {
    algorithm                   Algorithmidentifier,
    encryptedDigest             ENCRYPTED {HASHED {OfSignature}}}

RSASignature ::= INTEGER

DSASignature ::= SEQUENCE {     – for DSA and its variants
    r               INTEGER,
    s               INTEGER }
```

9.3.2 This section defines a data structure that contains an item (document) and one or more signatures on the item. This

structure contains the item and one or more signature structures. Each signature structure contains the signer's identifier, and indication of the signature algorithm, the digest of the document, and a set of attributes to be included in the signature computation (the signature attributes). All of these items are included in the signature computation. Additionally, one or more unsigned attributes (countersignatures) may be included.

9.3.3 The document being signed may be included in the **SignedData** structure that follows, in the **ContentInfo** field. Alternatively, the content may be stored separately, in which case the **SignedData** structure contains only the signatures and other relevant cryptographic information. This structure is taken from RFC 2630, which is in turn derived from PKCS #7. This specification places constraints on some of the fields in the CMS data structures.

```
SignedData ::= SEQUENCE {
    version         INTEGER { v1(1), v2(2), v3(3) },
    digestAlgorithms            SET OF AlgorithmIdentifier,
    contentinfo                 Contentinfo,
    certificates                IMPLICIT Certificate Set OPTIONAL,
    crls                  [1]   IMPLICIT CertificateRevocationLists
                                OPTIONAL,
    signatures      SEQUENCE OF Signerinfo }

ContentInfo :: = INSTANCE OF CONTENT-INFO ({SupportedContentTypes})

CONTENT-INFO ::= TYPE-IDENTIFIER

data CONTENT-TYPE ::= {OCTET STRING IDENTIFIED BY id-data }
signedData CONTENT-TYPE ::= { SignedData IDENTIFIED BY id-
signedData }

Certificate Set ::= SET OF CertificateChoice
    (WITH COMPONENTS {extendedCertABSENT, ... })

CertificateChoice ::= CHOICE {
    certificate     Certificate     –see below
    extendedCert [0] ExtendedCertificate,     – obsolete
    attributeCert  [1] AttributeCertificate }     – from X.509
```

9.3.4 The fields of type **SignedData** have the following meanings:

9.3.4.1 **version** is the syntax version number. It shall be 3 if attribute certificates are present, and 1 otherwise.

9.3.4.2 **digestAlgorithms** is a collection of message-digest algorithm identifiers. They may be any number of elements in the collection, not including zero. Each element identifies the message-digest algorithm (and any associated parameters) under which the content is digested for a signer. The collection is intended to list the message-digest algorithms employed by all of the signers, in any order, to facilitate one-pass signature verification. The message-digesting process is described 9.3.8.

9.3.4.3 **contentInfo** is the content that is signed. It can have any of the defined content types.

9.3.4.4 **certificates** is a set of X.509 public key and (optionally) attribute certificates. It is intended that the set be sufficient to contain public key certificate chains from a recognized "root" or "top-level certification authority" to all of the signers in the **signerInfos** field. There may be more certificates than necessary, and there may be certificates sufficient to contain chains from two or more independent top-level certification authorities. There may also be fewer certificates than necessary, if it is expected that those verifying the signatures have an alternate means of obtaining necessary certificates (for example, from a previous set of certificates). Note that PKCS #6 extended certificates shall not be used; equivalent functionality can be obtained using certificate extensions as discussed in 9.3.13.10. Attribute certificates may also be present to convey additional user privileges, but this version of this specification does not define any such privileges.

9.3.4.5 **crls** is a set of certificate-revocation lists. It is intended that the set contain information sufficient to determine whether or not the certificates in the certificates field are "hot listed," but such correspondence is not necessary. There may be more certificate-revocation lists than necessary, and there may also be fewer certificate-revocation lists than necessary.

9.3.4.6 **signerInfos** is a collection of per-signer information. There may be any number of elements in the collection, including zero.

9.3.5 Per-signer information is represented in the type **SignerInfo**:

```
SignerInfo ::= SEQUENCE {
    version             INTEGER { v1(1) },
    signer                          IssuerSerial,
    digestAlgorithm                 AlgorithmIdentifier,
    signedAttributes                SET OF Attribute,
    signatureAlgorithm              AlgorithmIdentifier,
    signature                       OCTET STRING,
    unsignedAttributes              SET OF Attribute }

IssuerSerial ::= SEQUENCE {
    issuer              Name,
    serial              CertificateSerialNumber }
```

9.3.6 The fields of type **SignerInfo** have the following meanings:

9.3.6.1 **version** is the syntax version number. It shall be 2 for this version of this specification.

9.3.6.2 **issuerSerial** specifies the signer's certificate (and thereby the signer's distinguished name and public key) by issuer distinguished name and issuer-specific serial number. This can be used to locate the certificate within the **certificates** field of the enclosing **SignedData** structure.

9.3.6.3 **digestAlgorithm** identifies the message-digest algorithm (and any associated parameters) under which the content and signed attributes (if present) are digested. It should be among those in the **digestAlgorithms** field of the enclosing **SignedData** value. The message-digesting process is described in 9.3.8 and 9.3.9.

9.3.6.4 **signedAttributes** is a set of attributes that are signed (authenticated) by the signer. The field is optional, but it shall be present if the content type of the **ContentInfo** value being signed is not *data*. If the field is present, it shall contain, at a minimum, two attributes: (*1*) A content-type attribute having as its value the content type of the **ContentInfo** value being signed. (*2*) A message-digest attribute, having as its value the message digest of the content (see 9.3.8 and 9.3.9). Other signature attribute types that might be useful here are also defined in PKCS #9.

9.3.6.5 **signatureAlgorithm** identifies the signature (and any associated parameters) under which the message digest and associated information are signed with the signer's private key. The signature process is described in 9.3.8.

9.3.6.6 **signature** is the result of signing the message digest and associated information with the signer's private key.

9.3.6.7 **unsignedAttributes** is a set of attributes that are not signed (authenticated) by the signer. The field is optional. The only currently defined unsigned attribute is the **countersignature** attribute.

9.3.7 The signing process is composed of two functions: message-digesting and the actual signature function.

9.3.8 The message-digesting process computes a message digest on either the content being signed or the content together with the signer's signature attributes. In either case, the initial input to the message-digesting process is the "value" of the content being signed. Specifically, the initial input is the contents octets of the DEF encoding of the content field of the **ContentInfo** value to which the signing process is applied. Only the contents octets of the DER encoding of that field are digested, not the identifier octets or the length octets.

9.3.9 The result of the message-digesting process (which is informally called the "message digest") depends on whether the **signedAttributes** field is present. When the field is absent, the result is just the message digest of the content. When the field is present, however, the result is the message digest of the complete DER encoding of the **Attributes** value contained in the **signedAttributes** field. (For clarity: The IMPLICIT [0] tag in the **signedAttributes** field is not part of the **Attributes** value. The **Attributes** value's tag is SET OF, and the DER encoding of the SET OF tag, rather than the IMPLICIT [0] tag, is to be digested along with the length and contents octets of the **Attributes** value. Since the **Attributes** value, the field is present, must contain as attributes the content type and the message digest of the content, those values are indirectly included in the result.

9.3.10 When the content being signed has content type *data* and the **signedAttributes** field is absent, then just the value of the data (the contents of a file) is digested. This has the advantage that the length of the content being signed need not be known in advance of the encryption process.

9.3.11 The input to the signature generation process includes the result of the message digest calculation process and the signer's private key. The details of the signature generation depend on the signature algorithm employed. The object identifier, along with any parameters, that specifies the signature algorithm employed by the signer is carried in the **signatureAlgorithm** field. The signature value generated by the signer is encoded as an OCTET STRING and carried in the signature field.

9.3.12 This section defines the contents of public key certificates and certificate revocation lists (CRLs).

```
CertificateInfo ::= SEQUENCE {
   version     [0]     Version DEFAULT v1,
   serialNumber            CertificateSerialNumber,
   signature               Algorithmidentifier,
   issuer                  Name, –    CA's name
   validity            Validity,
   subject                 Name,
   subjectPublicKeyInfo    SubjectPublicKeyInfo,
   issuerUniqueID          [1]     IMPLICIT UniqueIdentifier OPTIONAL,
   subjectUniqueID         [2]     IMPLICIT UniqueIdentifier OPTIONAL,
   extensions              Extensions OPTIONAL }

Certificate ::= SIGNED {CertificateInfo}

Version ::= INTEGER { v1(0), v2(1), v3(2) }

Name ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::= SET OF AttributeValueAssertion

AttributeValue Assertion ::= SEQUENCE {
   type     ATTRIBUTE.&id ({SupportedAttributes}),
   value    ATTRTIBUTE.&Syntax ({SupportedAttributes} {@type}) }
SubjectPublicKeyInfo ::= SEQUENCE {
   algorithm          AlgorithmIdentifier,
   entityPublicKey          BIT STRING }

Validity ::= SEQUENCE {
   notBefore    UTCTime,
   notAfter     UTCTime

CertificateSerialNumber ::= INTEGER

Extensions ::= SET OF Extension

Extension ::= SEQUENCE {
   extnId                  EXTENSION.&id ({ExtensionSet}),
critical                BOOLEAN DEFAULT FALSE,
extnValue                  OCTET STRING } – Contains a
–canonical encoding of a value of type &ExtnType for the
–extension object identified by extnId –

EXTENSION :: = CLASS
{
   &id                 OBJECT IDENTIFIER UNIQUE,
&ExtnType
}
WITH SYNTAX
{
SYNTAX              &ExtnType
IDENTIFIED BY &id
}
```

9.3.13 The certificate fields have the following meanings:

9.3.13.1 **version** is used to differentiate between versions of the certificate. Version 3 shall be used for this version of this specification.

9.3.13.2 **serialNumber** field uniquely identifies this certificate among all those issued by the same CA. The **serialNumber** shall never repeat for a given CA. The combination of issuer name and serial number uniquely identifies a certificate for purposes such as Certificate Revocation Lists.

9.3.13.3 **signature** identifies the algorithm (combination of digest and asymmetric signature algorithms) used to sign the certificate. It is added by the CA when creating the certificate.

9.3.13.4 **issuer** contains the distinguished name of the CA and is added by the CA when creating the certificate.

9.3.13.5 **validity** indicates the period during which a public/private key pair is valid.

9.3.13.6 **subject** is the name of the entity being certified. Names in certificates are defined as a sequence of components called relative distinguished names, each of which is an attribute type and value. The entity that is to be bound to the public key is contained in the **subject** field.

9.3.13.7 **subjectPublicKeyInfo** is the public key of the subject being certified.

9.3.13.8 **issuerUniqueID** is an optional field which uniquely identifies the subject being certified. This field is used in the case where the issuer name may be reused over time. (At any single point in time, the issuer name shall be unique, as discussed in 9.3.13.4. This does not preclude another entity using the name when the first entity no longer exists.)

9.3.13.9 **subjectUniqueID** is an optional field which uniquely identifies the subject being certified. This field is used in the case where the Distinguished Name of a subject may be reused over time.

9.3.13.10 **extensions** is an optional field in version 3 certificates which may contain additional information used during certificate verification or by applications. A standard set of generally useful extensions is being defined by ISO and ANSI; additional extensions may be defined by other groups. Critical extensions shall be recognized and processed by certificate-handling systems; unrecognized critical extensions will cause certificates to be considered invalid (for example, signature verification using such a certificate will fail). Unrecognized non-critical extensions may be ignored. The following extensions are useful for healthcare applications; syntax can be found in ANSI X9.57:

(1) Authority Key Identifier— Identifies CA key used to sign a certificate.

(2) Subject Key Identifier— Identifies user key used to sign a document.

(3) Key Usage— Constrains functions a key is used for (like signature or encryption).

(4) Certificate Policies and Policy Mappings— Allow domains to support different certification policies (for example, assurance levels).

(5) Alternative Names— Allows identification of issuer and subject by other name forms, such as e-mail addresses.

(6) Constraints— Allows a CA to constrain the length of a certification path, the namespace subordinate CAs may certify, and which policies may be present in a certification path.

9.3.14 This section defines the contents of certificate revocation lists (CRLs).

```
CertificateRevocationList ::= SIGNED {CRLInfo}

CRLInfo::=SEQUENCE {
    version                 INTEGER OPTIONAL,
    – only present if there are critical extensions
    signature                       AlgorithmIdentifier,
    issuer                          Name,
    issuerUID [0] UniqueIdentifier OPTIONAL,
    lastUpdate              UTCTime,
    nextUpdate                      UTC Time OPTIONAL,
    revokedCertificates             SEQUENCE OF CRLEntry DEFAULT {},
    extensions[1] Extensions OPTIONAL }

CRLEntry ::- SEQUENCE {
    certificate                     CertificateSerialNumber,
    revocationDate                  UTCTime,
    extensions                      Extensions OPTIONAL }
```

9.3.15 The CRL fields have the following meanings:

9.3.15.1 **version** is used to differentiate between versions of the CRL. For compatibility with earlier versions of the CRL, this field is only present if critical extensions are present in the CRL. Version 2 is recommended, but not required, for use with this specification.

9.3.15.2 **signature** identifies the algorithm (combination of digest and asymmetric signature algorithms) used to sign the certificate. It is added by the CA when creating the certificate.

9.3.15.3 **issuer** contains the distinguished name of the CA issuing the CRL.

9.3.15.4 **issuerUniqueID** is an optional field which uniquely identifies the issuer. This field is used in the case where the issuer name may be reused over time.

9.3.15.5 **lastUpdate** indicates the date and time this CRL was issued by the CA.

9.3.15.6 **nextUpdate** indicates the date and time the next scheduled CRL will be issued. Note this does not preclude the CA from issuing CRLs prior to that time, if security policy so dictates.

9.3.15.7 **revokedCertificates** is a list of the revoked certificates, each element of which is a **CRLEntry**. Revoked certificates remain on the CRL till they expire.

9.3.15.8 **extensions** contain optional additional information associated with the CRL. The following extensions are useful to partition CRLs into more manageable sizes:

(1) CRL Number—a serial number for CRLs, to detect the need to retrieve missing CRLs.

(2) Issuing Distribution Point—Allows CRLs to be partitioned by revocation reason. A certificate extension indicating the distribution point(s) for each certificate is also defined.

(3) Delta CRL Indicator—Indicates the CRL contains only updates since the reference base URL.

9.3.16 The CRL entry fields have the following meanings:

9.3.16.1 **certificate** contains the serial number of the revoked certificate.

9.3.16.2 **revocationDate** contains the date and time the certificate was revoked.

9.3.16.3 **extensions** contains optional additional information associated with this particular revoked certificate. Useful extensions include:

(1) Revocation Reasons—Some reasons, such as key compromise, require different handling than other reasons;

(2) Invalidity Date—Date prior to which the key was not compromised. This allows verification of signatures created before this date, even after the certificate is revoked.

9.4 *Signature Attributes*:

9.4.1 This section specifies the signature attributes defined in Guide E 1762.

```
signature-purpose ATTRIBUTE ::= {
        WITH ATTRIBUTE SYNTAX       signaturePurpose
        SINGLE VALUE                TRUE
        ID                          id-signature-purpose }

SignaturePurpose ::= OBJECT IDENTIFIER

–OIDs for the purposes defined in E 1762; see Annex A1 for values
        id-purpose-author
        id-purpose-co-author
        id-purpose-co-participant
        id-purpose-transcriptionist
        id-purpose-verification
        id-purpose-validation
        id-purpose-consent
```

ANNEX

(Mandatory Information)

A1. OBJECT IDENTIFIERS

A1.1 This annex defines the object identifiers used for the attributes and document types defined in this specification.

```
–roots for OIDs
pkcs-9 OBJECT IDENTIFIER ::=
      { iso(1) member-body(2) US(840)rsadsi)(113549) pkcs(1) 9 }
attribute OBJECT IDENTIFIER ::=
      { iso(1) identified-organization(3) oiw(14) secsig(3) attribute(4) }

–new OIDs are registered under the E31.20 arc
e31-20 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) 10065 }
ps100 OBJECT IDENTIFIER ::= { e31-20 1 }
docattr OBJECT IDENTIFIER ::= { ps100 10 }
sigattr OBJECT IDENTIFIER ::= { ps100 11 }
sigpurpose OBJECT IDENTIFIER ::= { ps100 12 }
datatype OBJECT IDENTIFIER ::= { ps100 13 }
format OBJECT IDENTIFIER ::= { ps100 14 }

ID ::= OBJECT IDENTIFIER

–document attributes
id-transaction-type ID ::= { attribute 9 }
id-location ID ::= { attribute 10 }
id-amendment-to ID ::= { docattr 1 }
id-patient-id ID ::= { docattr 2 }
id-event-id ID ::= {docattr 3 }
id-data-type ID ::= { docattr 4 }
id-data-format ID ::= { docattr 5 }
id-orig-org ID ::= { docattr 6 }
id-event-time ID ::= { docattr 7 }
id-creation-time ID ::= { docattr 8 }
id-motif-time ID ::= { docattr 9 }
id-access-time ID ::= { docattr 10 }
id-doc-identifier ID ::= { docattr 11 }

–Attributes required for SignedData
content-type ::= {OBJECT IDENTIFIER IDENTIFIED BY id-content-type }
message-digest ::= { OCTET STRING IDENTIFIED BY id-message-digest }
id-content-type ::= { pkcs-9 3 }
id-message-digest ::= { pkcs-9 4 }

–OIDs for content types
pkcs-7 OBJECT IDENTIFIER ::=
      { iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) 7 }
id-date OBJECT IDENTIFIER ::= { pkcs-7 1}
id-signedData OBJECT IDENTIFIER ::= { pkcs-7 2 }

–OIDs for signature attributes and unsigned attributes
id-signature-purpose ID ::= { sigattr 1 }
id-signing-time ID ::= { sigattr 2 }
id-biometric-info ID ::= { sigattr 3 }
id-signature-reason ID ::= { sigattr 4 }
id-machine-id ID ::= { sigattr 5 }
id-annotation ID ::= { sigattr 6 }
id-countersignature ID := { pkcs-9 6 }

–signature purposes
id-purpose-author ID ::= { sigpurpose 1 }
id-purpose-co-author ID ::= { sigpurpose 2 }
id-purpose-co-participated ::= { sigpurpose 3 }
id-purpose-transcriptionist ::= { sigpurpose 4 }
id-purpose-verification ::= { sigpurpose 5 }
id-purpose-validation ::= { sigpurpose 6 }
id-purpose-consent ::= { sigpurpose 7 }
```

```
id-purpose-witness ::= { sigpurpose 8 }
id-purpose-event-witness ::= { sigpurpose 9 }
id-purpose-identity-witness ::= { sigpurpose 10 }
id-purpose-consent-witness ::= { sigpurpose 11 }
id-purpose-interpreter ::= { sigpurpose 12 }
id-purpose-review ::= { sigpurpose 13 }
id-purpose-source ::= { sigpurpose 14 }
id-purpose-addendum ::= { sigpurpose 15 }
id-purpose-administrative ::= { sigpurpose 16 }
id-purpose-timestamp ::= { sigpurpose 17}
```

APPENDIXES

(Nonmandatory Information)

X1. RSA AND RELATED SIGNATURE ALGORITHMS

X1.1 This appendix describes the RSA signature algorithm and its variants. It recommends particular variants for particular uses.

X1.2 RSA is based on the difficulty of factoring large numbers. A user generates a public/private key pair as follows:

X1.2.1 Choose two large primes, p and q. They should be the same length, between 384 and 1024 bits.

X1.2.2 Choose an odd exponent e, which is relatively prime to $(p\text{-}1)(q\text{-}1)$.

X1.2.3 Compute $d = e^{-1} \bmod ((p\text{-}1)(q\text{-}1))$.

X1.2.4 The public key is then (e, n), and the private key is (d, n).

X1.3 To sign a message, it is digested, and the digest is padded to the size of the modulus. PKCS #1 specifies a simple padding mechanism, while ISO 9796 and ANSI X9.31 specify a more complex mechanism. If m is the padded digest, the signature is simply:

$$s = m^d \bmod n \tag{X1.1}$$

X1.4 To verify the signature, digest the received message, and compute:

$$m' = s^e \bmod n \tag{X1.2}$$

Extract the digest from m' (a padded digest) and compare to the computed digest. The signature is valid if the two digests are equal.

X1.5 RSA is the most widely deployed digital signature algorithm. A large number of products are available which implement the PKCS #1 padding method. There are also some products supporting ISO 9796-2 and ANSI X9.31 padding. From a performance point of view, signature verification is much faster than competing algorithms. RSA is patented in the United States; its use requires a license.

X1.6 RSA using either PKCS #1 or ISO 9796-2 is recommended for general use, with a modulus length between 1024 and 2048 bits.

X2. DSA AND RELATED SIGNATURE ALGORITHMS

X2.1 This appendix describes the DSA signature algorithm and its variants. It recommends particular variants for particular uses. These algorithms are based on the difficulty of computing discrete logarithms. They are all variants of the ElGamal algorithm.

X2.2 The ElGamal algorithm uses a common modulus p, which is a large prime, and a common generator g between 2 and p-2. Each user chooses a private key x, and computes her public key y as $g^x \bmod p$.

X2.2.1 To sign a message, its digest m is computed, a random number k between 2 and p-2 is generated, and the following calculations are performed:

$$r = g^k \bmod p \tag{X2.1}$$

$$s = (m - xr)k^{-1} \bmod p \tag{X2.2}$$

The signature is the pair (r, s). Note this signature is twice the length of p.

X2.2.2 To verify a signature, compute the digest of the received message m'. The signature is valid if:

$$g^{m'} = y^r r^s \bmod p \tag{X2.3}$$

X2.3 The NIST Digital Signature Algorithm (DSA) was designed by David Kravitz, who was then at NSA. g is a generator of a multiplicative subgroup of a GF(p) of order q, where q is a prime number which divides p-1. The private key x is between 2 and q-2, and the public key y is computed as in X2.2. DSA is described in ANSI X9.30.

X2.3.1 Signature generation uses the message digest m; k is between 1 and q-1. The following calculations are performed:

$$r = (g^k \bmod p) \bmod q \tag{X2.4}$$

$$r = (m + xr)k^{-1} \bmod q \tag{X2.5}$$

X2.3.2 Signature verification consists of checking that:

$$r = (g^{mw} y^{rw} \bmod p) \bmod q, \text{ where } w = s^{-1} \tag{X2.6}$$

X2.4 DSA signatures may also be computed using the group of points on an elliptic curve rather than the multiplicative subgroup of order *q* described in X2.3. This variant is described in ANSI X9.62. The computations are as described in Eq X2.4-X2.6. Operations map to elliptic curves as follows:

X2.4.1 The generator *g* is a point on the curve.

X2.4.2 The subgroup modulus *q* maps to the order of the generator point.

X2.4.3 Multiplication modulo *p* maps to addition of points.

X2.4.4 Exponentiation maps to scalar multiplication of points.

X2.4.5 Use of elliptic curves greatly reduces the size of the numbers involved. For example, a 1024–bit key in DSA can be replaced by a 155-bit key in ECDSA. This leads to savings in computational requirements (although the underlying operations are more complex), as well as storage and bandwidth requirements.

X2.5 The signature algorithms in 3.1.2-3.1.9 all provide the same capabilities. Given that DSA is an ANSI and U.S. government standard, it is recommended for general use. ECDSA is recommended for those cases where storage and bandwidth are limited (for example, smart cards). DSA is patented, but the government has declared that it can be used royalty-free.