Standard ECMA-235
March 1996

ECMA

Standardizing Information and Communication Systems

The ECMA GSS-API Mechanism

Phone: +41 22 849.60.00 - **Fax:** +41 22 849.60.01 - **URL:** http://www.ecma.ch - **Internet:** helpdesk@ecma.ch

Standard ECMA-235
March 1996

ECMA

Standardizing Information and Communication Systems

The ECMA GSS-API Mechanism

Phone: +41 22 849.60.00 - **Fax:** +41 22 849.60.01 - **URL:** http://www.ecma.ch - **Internet:** helpdesk@ecma.ch

Gino Lauri - ECMA-235.DOC - 20/03/96 11:50

Brief History

ECMA, ISO and ITU-T are working on standards for distributed applications in an open system environment. Security in general and authentication and distributed access control in particular are major concerns in information processing.

In July 1988, ECMA TR/46, "Security in Open Systems - A Security Framework", was published. In December 1989, based on the concepts of this framework, ECMA-138, "Security in Open Systems - Data Elements and Service Definitions", was produced. It defines a set of Security Services for use in the Application Layer of the ISO OSI Reference Model.

In December 1994, the first edition of Standard ECMA-219 was published. Based on this earlier work, it describes a model for distributed authentication and access control in which a trusted third party, the Authentication and Privilege Attribute Application (APA-Application) and related key distribution functions are used to authenticate human and software entities, provide them with the privileges they need for access control purposes and provide the means of protection of these privileges in interchange.

Over this period also, the Internet Engineering Task Force (IETF) and other de facto and de jure standards organisations have been developing a standard general interface through which a security infrastructure such as that described in [ECMA-219] can be exercised by application clients and servers. It has been designed so that callers do not need to know the details of the underlying infrastructure, or even whether it is provided by [ECMA-219] services or other infrastructure designs. This interface is the Generic Security Services Application Programming Interface, or GSS-API ([GSS-API]).

This ECMA standard follows on from [ECMA-219], showing how the security services described there can be used underneath the GSS-API by application clients and servers. It describes the interface calls supported, the success and error responses that can be returned, and the format and content of the data tokens exchanged between the client and the server.

In order to implement Privilege Attribute based access control features for distributed open applications using the GSS-API, this Standard also defines support functions that have to be used in addition to the standard GSS-API function set.

[KRB5GSS] and [SPKM] also define ways of supporting the GSS-API, and some of the data constructs defined there are also used here.

The Standard is based on the practical experience of ECMA member Companies. It is oriented towards urgent and well understood needs.

This ECMA Standard has been adopted by the ECMA General Assembly in March 1996.

Table of contents

1 Introduction

1.1 Scope

Standard ECMA-219 defines services, data elements and operations for authentication, Privilege Attribute and key distribution applications (the APA-Application).

Following on from [ECMA-219], this Standard ECMA-235defines the syntax of the tokens that enable distributed applications implementing the APA-Application and related data elements specified in Standard ECMA-219 to interwork. The tokens defined in this Standard are :

- Tokens for Security Association establishment
- An error token for communicating a failure to establish a Security Association
- Tokens for message protection
- A token for Security Association deletion

In order to provide a basic set of implementation options, this Standard also defines some key distribution schemes based on symmetric and asymmetric cryptographic technologies. These include specification of the encryption algorithms and methods to be used.

The tokens are intended for use through the Generic Security Service API (GSS-API) as defined in [GSS-API]. This Standard defines minor status returns that are returned by the GSS-API when a GSS-API conformant implementation is used to generate and validate the tokens.

In order to implement Privilege Attribute based access control features for distributed open applications using the GSS-API, this Standard also defines support functions that have to be used in addition to the standard GSS-API function set.

1.2 Field of application

The field of application of this ECMA Standard is the design, implementation and interworking of security modules that make use of the APA-Application as defined in [ECMA-219]. They define an implementation of the "ECMA GSS-API mechanism".

1.3 Requirements to be satisfied

Requirements for secure distributed environments have led to specifications of security services such as [ECMA-219], [Kerberos] and [SPKM]. Each of these defines what is known in [GSS-API] as a "security mechanism".

The [ECMA-219] mechanism defines security services and data elements required to secure distributed applications. However, in order to achieve interworking between normal application servers using these security services , the syntax of the security tokens to be exchanged between the application servers themselves needs to be defined.

[GSS-API] specifies an interface that is independent of the underlying supporting security mechanism, but through which mechanism-specific security tokens can be exchanged. The GSS-API is intended to be used by implementors of distributed secured applications.

The GSS-API provides functions to implement identity based access control policies, but it does not provide support functions to handle in a generic way Privilege Attributes for access control purposes. Neither does it provide for the control of delegation. This standard therefore specifies such support functions.

1.4 Conformance

There are a number of types of conformance to this Standard as follows :

Type 1 Support functions conformance

The implementation shall be conformant to [GSS-API], with the addition of the ECMA mechanism support functions defined in clause 9. Any minor status returns must be from the set defined in clause 8. This type of conformance is in support of application portability, and does not demand that the underlying GSS-API mechanism is the ECMA one.

Type 2 Security Association level context token conformance

The implementation shall support at least one mechanism option of the ECMA mechanism Security Association establishment, deletion, and error tokens defined in clause 4.1 to 4.4 and 4.6 Any minor status returns must be from the set defined in clause 8. This type of conformance is in support of interoperability, and does not require support for the GSS-API.

Type 3 Message level token conformance

The implementation shall be Type 2 conformant, and also provide an implementation of the ECMA mechanism message protection tokens defined in clause 4.5 Any minor status returns must be from the set defined in clause 8.

Type 4 Full ECMA GSS-API mechanism conformance

This is achieved if both Type 1 and Type 3 conformance are achieved

1.5 Overview and document structure

The standard described in [ECMA-219] defines specific service interfaces to security services supporting the provision of authentication, key establishment, data integrity, data confidentiality and access control information. Although the scope of that standard does not encompass the specification of how to establish Security Associations with productive application servers, it does assume and describe a model for these exchanges. The combined model and standard is defined as the ECMA mechanism. This document describes how the generic ECMA mechanism is to be exercised through the GSS-API to form the ECMA GSS-API Mechanism. Contents of specific clauses are:

Clauses 2 and 3:	These contain the usual references and definitions respectively.
Clause 4:	Describes the token formats exchanged between GSS-API peers using the ECMA GSS-API mechanism.
Clause 5:	Defines specific key distribution schemes within the framework laid down in [ECMA-219]. It gives detailed syntax and semantics for these schemes.
Clause 6:	Describes the use of cryptographic algorithms in the ECMA GSS-API mechanism.
Clause 7:	Describes the ways in which OBJECT IDENTIFIERS are used to nominate particular specific ECMA GSS-API mechanism types, including the choice of cryptographic algorithms themselves.
Clause 8:	Describes the GSS-API minor status codes that can be returned by the ECMA GSS-API mechanism. See also annex E.
Clause 9:	Defines additional GSS-API support functions needed to enable PAC attribute and control information to be set and exploited by GSS-API callers. It also defines some specific attribute types.
Clause 10:	Explains the relationship between this standard and other standards.
Annex A:	Contains normative formal ASN.1 definitions of ASN.1 defined in this standard.
Annex B:	Contains normative formal ASN.1 definitions of ASN.1 also used in [SPKM].
Annex C:	Contains normative formal ASN.1 definitions of ASN.1 also used in [Kerberos].
Annex D:	Maps the minor status codes given in clause 8 onto the relevant error values defined in [ECMA-219].
Annex E:	Expands the imported ASN.1 constructs (for information purposes).

2 References

2.1 Normative references

ECMA-219	ECMA-219, Authentication and Privilege Attribute Application with related key distribution functions
GSS-API	1. Internet RFC 1508 Generic Security Service API (J. Linn, September 1993)
	2. X/Open P308 Generic Security Service API (GSS-API) Base
	3. Internet RFC 1509 "Generic Security Service API: C-Bindings"

Kerberos	Internet RFC 1510 The Kerberos Network Authentication Service (V5) (J. Kohl and C. Neumann, September 1993)
ISO 10745	ISO 10745, Upper Layers Security Model
ISO/IEC 9594-2	ISO/IEC 9594-2, Information Processing Systems - Open Systems Interconnection - The Directory - Part 2: Information Framework (X.501)
ISO/IEC 9594-8	ISO/IEC 9594-8, Information Processing Systems - Open Systems Interconnection - The Directory - Part 8: Authentication Framework (X.509)

2.2 Informative references

KERB5GSS	draft-ietf-cat-kerb5gss-03 The Kerberos Version 5 GSS-API Mechanism (J. Linn, September 1995)
SPKM	draft-ietf-cat-spkmgss-04: The Simple Public-Key GSS-API Mechanism (C. Adams, May 1995)
SNEGO	draft-ietf-cat-snego-00 Simple GSS-API Negotiation Mechanism (Eric Baize and Denis Pinkas, July 1995)

3 Definitions

3.1 Imported definitions

The following terms are used with the meaning defined in [ECMA-219] :

access identity
attribute set reference
Audit Identity
basic key
delegate
dialogue key
External Control Value
Privilege Attribute Certificate
target
Target AEF
target key block

The following terms are used with the meaning defined in [GSS-API]

acceptor
initiator
channel bindings
context acceptor
context Initiator
credentials
GSS-API token
mechanism type
quality of protection

The following terms are used with the meaning defined in [ISO 10745]:

Security Association

3.2 New Definitions

3.2.1 Security Context

Security information that represents, or will represent a Security Association to an initiator or acceptor that has formed, or is attempting to form such an association.

3.2.2 Generic Security Mechanism

A generic security mechanism identifies a class of support functions, data structures and protocols from which specific security mechanism options can be derived.

3.2.3 Security Mechanism Options

A security mechanism option identifies for a generic security mechanism, a specific choice of support functions, data structures and protocols required to an initiator and an acceptor in order to establish and use security contexts

3.2.4 Primary Principal Identifier (PPID)

An arbitrary identifier of the original source of a request for a PAC. It is a specific example of a value used in the "primary principal qualification" protection method. The PPID is used to prevent a PAC from being stolen or delegated. See [ECMA-219] for an explanation of the protection method.

3.3 Acronyms

ACI	Access Control Information
AEF	Access Enforcement Function
CA	Certification Authority
DES	Data Encryption Standard
ECV	External Control Value
GSS-API	Generic Security Service Application Program Interface
KD-	Key Distribution (server)
PAC	Privilege Attribute Certificate
PPID	Primary Principal Identifier
RSA	Rivest Shamir Adleman
SA	Security Association
SPKM	Simple Public Key Mechanism

4 Token formats

This clause describes protocol-visible characteristics of the GSS-API as implemented above the ECMA mechanism. Succeeding sub-clauses define the syntax of tokens exchanged between GSS-API peers for Security Association management.

4.1 Token framings

Following [GSS-API], tokens are enclosed within framing as follows:

```
Token ::=
    [APPLICATION 0] IMPLICIT SEQUENCE {
      thisMech              MechType,   -- the OBJECT IDENTIFIER specified below
      innerContextTokenANY DEFINED BY thisMech }
```

The ECMA mechanism type is identified by an OBJECT IDENTIFIER with value:

```
235{ generic-ecma-mech (y) (z) }
```

Where:

```
generic-ecma-mech ::=
    { iso(1) identified-organisation(3) icd-ecma(0012) standard(0) ecma-gss-api (235)
    generic-ecma-mech (4)}
```

See clause 7 for the values of y and z that have been defined in this Standard.

The above GSS-API framing shall be applied to all tokens emitted by the ECMA GSS-API mechanism, including context-establishment tokens, per-message tokens, and context-deletion token.

The innerContextToken field of context establishment tokens for the ECMA GSS-API mechanism will consist of an ECMA token (InitialContextToken, TargetResultToken, ErrorToken) containing a token identifier (tokenId) field having the value 01 00 (hex) for InitialContextToken, 02 00 (hex) for TargetResultToken, and 03 00 (hex) for ErrorToken. These are defined to be:

InitialContextToken	sent by the initiator to a target, to start the process of establishing a Security Association. Returned by the GSS_Init_sec_context call.
TargetResultToken	sent to the initiator by the target following receipt of an InitialContext Token. Returned by the GSS_Accept_sec_context call.
ErrorToken	sent by target on detection of an error during Security Association establishment. Returned by either the GSS_Init_sec_context call or the GSS_Accept_sec_context call.

The innerContextToken field of per-message tokens for the ECMA GSS-API mechanism will consist of an ECMA token (MICToken, WrapToken) containing a tokenId field having the value 01 01 (hex) for MICToken, and 02 01 (hex) for WrapToken. These are defined to be:

MICToken	sent either by the initiator or the target to verify the integrity of the user data sent separately. Returned by GSS_GetMIC.
WrapToken	sent either by the initiator or the target. Encapsulates the input user data (optionally encrypted) along with integrity check values. Returned by GSS_Wrap.

The innerContextToken field of context-deletion token for the ECMA GSS-API mechanism will consist of an ECMA token (ContextDeleteToken) containing a tokenId field having the value 03 01 (hex). This is defined to be:

ContextDeleteToken	sent either by the initiator, or the target to release a Security Association. Returned by GSS_Delete_sec_context.

4.2 InitialContextToken format

This construct provides for the carrying of the following Security Association information:

- replay protection,
- keying information for establishing both the key for protecting the exchange of context tokens, including this one, and keys for later use in protecting user data exchanges,
- access control information (ACI) used to determine what access is to be granted by the target to the initiator of the Security Association,
- information to control the applicability of the ACI (e.g. whether delegation is permissible, or identifying for which targets the access rights are valid),
- information to protect the ACI from being tampered with or stolen,
- accountability information for use in audit trails.

```
InitialContextToken ::=  SEQUENCE    {
    ictContents      [0]     ICTContents,
    ictSeal          [1]     Seal   -- Imported from [ECMA-219] (see annex E)
                                                                    }
```

ictContents

Body of the initial context token

ictSeal

Seal of ictContents computed with the integrity dialogue key. Only the sealValue field of the Seal data structure is present. The cryptographic algorithms that apply are specified by integDKUseInfo in the dialogueKeyBlock field of the initial context token.

```
ICTContents ::= SEQUENCE {
    tokenId             [0]  INTEGER,       -- shall contain X'0100'
    SAId                [1]  OCTET STRING,
    targetAEFPart       [2]  TargetAEFPart,
    targetAEFPartSeal   [3]  Seal,  -- Imported from [ECMA-219] (see annex E)
    contextFlags        [4]  BIT STRING{
                                 delegation          (0),
                                 mutual-auth         (1),
                                 replay-detect       (2),
                                 sequence            (3),
                                 conf-avail          (4),
                                 integ-avail         (5)
                                    }
    utcTime             [5]  UTCTime        OPTIONAL,
    usec                [6]  INTEGER        OPTIONAL,
    seq-number          [7]  INTEGER        OPTIONAL,
    initiatorAddress    [8]  HostAddress    OPTIONAL,   -- imported from [Kerberos]
    targetAddress       [9]  HostAddress    OPTIONAL
                                                        -- used as channel bindings    }
```

tokenId

Identifies the initial-context token. Its value is 01 00 (hex)

SAId

A random number for identifying the Security Association being formed; it is one which (with high probability) has not been used previously. This random number is generated by the initiator ECMA GSS-API implementation and processed by the target GSS-API implementation as follows :

- If no targetResultToken is expected, the SAId value is taken to be the identifier of the Security Association being established (if this is unacceptable to the target, then an error token with etContents value of gss_ecma_s_sg_sa_already_established must be generated).
- If a targetResultToken is expected, the target generates its random number and concatenates it to the end on the initiator's random number. The concatenated value is then taken to be the identifier of the Security Association being established.

targetAEFPart

Part of the initial-context token to be passed to the target access enforcement function.

targetAEFPartSeal

Seal of the targetAEFPart computed with the basic key. Only the sealValue field of the Seal data structure is present. The cryptographic algorithms that apply are specified by algorithm profile in the ECMA mechanism option (see clauses 6 and 7).

contextFlags

Combination of flags that indicates context-level functions requested by the GSS-API initiator implementation.

delegation when set to 0, indicates that the initiator explicitly forbids delegation of the PAC in the targetAEFPart.

mutual-auth indicates that mutual authentication is requested.

replay-detect	indicates that replay detection features are requested to be applied to messages transferred on the established Security Association.
sequence	indicates that sequencing features are requested to be enforced for messages transferred on the established Security Association.
conf-avail	indicates that a confidentiality service is available on the initiator side for the established Security Association.
integ-avail	indicates that an integrity service is available on the initiator side for the established Security Association.

utcTime

The initiator's UTC time.

usec

Microsecond part of the initiator's time stamp. This field, along with utcTime, is used to specify a precise time stamp

seq-number

When present, specifies the initiator's initial sequence number. Otherwise, the default value of 0 is to be used as an initial sequence number.

initiatorAddress

Initiator's network address part of the channel bindings. This field is only present when channel bindings are transmitted by the GSS-API caller to the ECMA GSS-API implementation.

targetAddress

Target's network address part of the channel bindings. This field is only present when channel bindings are transmitted by the GSS-API caller to the ECMA GSS-API implementation.

```
TargetAEFPart ::= SEQUENCE  {
    pacAndCVs           [0] SEQUENCE OF CertandECV OPTIONAL,
    targetKeyBlock      [1]     TargetKeyBlock,     -- see clause 5 for schemes supported
    dialogueKeyBlock    [2]     DialogueKeyBlock, -- Imported from [ECMA-219] (see annex E)
    targetIdentity      [3]     Identifier,         -- Imported from [ECMA-219] (see annex E)
    flags               [4]     BIT STRING{
                                        delegation          (0)
                                    }
                            }
```

NOTE 1
ECMA validity philosophy is that individual inner certificates have validity of their own, and that it is not sensible to have an overall separately specified validity period for the whole context.

NOTE 2
ECMA does not permit the target to opt for a shorter validity time than that specified by the initiator. If it wants to cut off the context earlier it just does it, returning an appropriate error.

pacAndCVs

The ACI to be used for this Security Association. This field is not present when the association does not require any ACI. This field can contain several PACs in delegation schemes where the Security Association requires not only the initiator PAC to be present, but also delegate PACs.

targetKeyBlock

The targetKeyBlock carrying the basic key to be used for the Security Association being established.

dialogueKeyBlock

A dialogue key block used by the Target AEF along with the basic key to establish an integrity dialogue key and a confidentiality dialogue key for per-message protection over the Security Association being established.

targetIdentity

The identity of the intended target of the Security Association. Used by the Target AEF to validate the PAC. Can also be used by the Target AEF to help protect the delivery of dialogue keys.

flags

flags required by the Target AEF for its validation process. Only contains a delegation flag, the value of which is the same as the value of delegation flag in contextFlag field of ictContents. When the flag is set, all ECVs sent in pacAndCVs are made available to the target. Other bits are reserved for future use.

4.3 TargetResultToken

This token is returned by the acceptor if the mutual-req flag is set in the InitialContext Token. It serves to authenticate the acceptor to the initiator, since only the genuine acceptor could derive the integrity dialogue key needed to seal the TargetResultToken.

```
TargetResultToken ::=  SEQUENCE    {
    trtContents     [0] TRTContents,
    trtSeal         [1] Seal                                    }

TRTContents ::= SEQUENCE {
    tokenId         [0]     INTEGER,        -- shall contain X'0200'
    SAId            [1]     OCTET STRING,
    utcTime         [5]     UTCTime         OPTIONAL,
    usec            [6]     INTEGER         OPTIONAL,
    seq-number      [7]     INTEGER         OPTIONAL
                            }
```

NOTE
There is no field for returning certification data here. This is because any such data that may be required is assumed to be returned at the conclusion of mechanism negotiation.

trtContents

This contains only administrative fields, identifying the token type, the context and providing exchange integrity.

seq-number

When present, specifies the acceptor's initial sequence number, otherwise, the default value of 0 is to be used as an initial sequence number.

The other administrative fields are as described in clause 4.2.

trtSeal

Seal of trtContents computed with the integrity dialogue key. Only the sealValue field of the Seal data structure is present. The cryptographic algorithms that apply are specified by integDKUseInfo in the dialogueKeyBlock field of the initial context token.

4.4 ErrorToken

```
ErrorToken ::=  {
    tokenType       [0]     OCTET STRING VALUE X'0400',
    etContents      [1]     ErrorArgument,
                }
```

etContents

Contains the reason for the creation of the error token. The different reasons are given as minor status return values. Clause 8 describes these in more detail, and annex D maps them onto the error returns of [ECMA-219].

```
ErrorArgument ::= ENUMERATED  {
    gss_ecma_s_sg_server_sec_assoc_open                    (1),
    gss_ecma_s_sg_ incomp_cert_syntax                      (2),
    gss_ecma_s_sg_ bad_cert_attributes                     (3),
    gss_ecma_s_sg_ inval_time_for_attrib                   (4),
    gss_ecma_s_sg_ pac_restrictions_prob                   (5),
    gss_ecma_s_sg_ issuer_problem                          (6),
    gss_ecma_s_sg_ cert_time_too_early                     (7),
    gss_ecma_s_sg_ cert_time_expired                       (8),
    gss_ecma_s_sg_ invalid_cert_prot                       (9),
    gss_ecma_s_sg_ revoked_cert                            (10),
    gss_ecma_s_sg_ key_constr_not_supp                     (11),
    gss_ecma_s_sg_ init_kd_server_ unknown                 (12),
    gss_ecma_s_sg_init_unknown                             (13),
    gss_ecma_s_sg_alg_problem_in_dialogue_key_block        (14),
    gss_ecma_s_sg_no_basic_key_for_dialogue_key_block      (15),
    gss_ecma_s_sg_key_distrib_prob                         (16),
    gss_ecma_s_sg_invalid_user_cert_in_key_block           (17),
    gss_ecma_s_sg_unspecified                              (18),
    gss_ecma_s_sg_invalid_token_format                     (19)
                                    }
```

4.5 Per Message Tokens

The syntax of the Per Message Token has the same general structure for both MIC and Wrap tokens:

```
PMToken ::=  SEQUENCE    {
    pmtContents      [0]    PMTContents,
    pmtSeal          [1]    Seal
                            }

PMTContents ::= SEQUENCE {
    tokenId          [0]    INTEGER,
    SAId             [1]    OCTET STRING,
    seq-number       [2]    INTEGER                          OPTIONAL,
    userData         [3]    CHOICE {
                                    plaintext       BIT STRING,
                                    ciphertext      OCTET STRING
                                        }                    OPTIONAL,
    directionIndicator [4]  BOOLEAN                          OPTIONAL
                             }
```

pmtContents

tokenId

Identifies the type of per message token.

SAId

See clause 4.2 for a description of this field.

seq-number

This field must be present if replay detection or message sequencing have been specified as being required at Security Association initiation time. The field contains a message sequence number whose value is incremented by one for each message in a given direction, as specified by directionIndicator. The first message sent by the initiator following the InitialContextToken shall have the message sequence number specified in that token, or if this is missing, the value 0. The first message returned by the acceptor shall have the message sequence number specified in the TargetReplyToken if present, or failing this, the value 0.

The receiver of the token will verify the sequence number field by comparing the sequence number with the expected sequence number and the direction indicator with the expected direction indicator. If the sequence number in the token is higher than the expected number, then the expected sequence number is adjusted and GSS_S_GAP_TOKEN is returned. If the token sequence number is lower than the expected number, then the expected sequence number is not adjusted and GSS_S_DUPLICATE_TOKEN or GSS_S_OLD_TOKEN is returned, whichever is appropriate. If the direction indicator is wrong, then the expected sequence number is not adjusted and GSS_S_UNSEQ_TOKEN is returned

userData

See specific token type narratives below.

directionIndicator

Present if seq-number is specified. FALSE indicates that the sender is the context initiator, TRUE that the sender is the target.

pmtSeal

See specific token type narratives below.

4.5.1 MICToken

Use of the GSS_Get_MIC() call yields a per-message token, separate from the user data being protected, which can be used to verify the integrity of that data as received. The token and the data may be sent separately by the sending application and it is the receiving application's responsibility to associate the received data with the received token. The syntax of the token is:

```
MICToken ::= PMToken
```

The overall structure and field contents of the token are described in clause 4.5. Fields specific to the MICToken are:

tokenId

Shall contain 01 01 (hex) to identify the token as a MICToken

userData

Not present for MIC Tokens.

pmtSeal

Computed over a DER encoding of pmtContents, but as if the data to be protected were present as plaintext in the userData field. The result binds the data to the entire plaintext header, so as to minimize the possibility of malicious splicing.

4.5.2 WrapToken

Use of the GSS_Wrap() call yields a token which encapsulates the input user data (optionally encrypted) along with associated integrity check values. The token emitted by GSS_Wrap() consists of an integrity header followed by a body portion that contains either the plaintext data (if conf_alg = NULL) or encrypted data. The syntax of the token is:

```
WrapToken ::=  PMToken
```

The overall structure and field contents of the token are described in clause 4.5. Fields specific to the WrapToken are:

tokenId

Shall contain 02 01 (hex) to identify the token as a WrapToken

userData

Present either in plain text form (the choice is plaintext), or encrypted (choice ciphertext). If the data is encrypted, the encryption is performed using the Confidentiality Dialogue Key, and as in [Kerberos], an 8-byte random confounder is first prepended to the data to compensate for the fact that an IV of zero is used for encryption.

wtSeal

The Checksum is calculated over the DER encoding of the pmtContents field, including the userData. However if the userData field is to be encrypted, the seal value is computed prior to the encryption.

4.6 ContextDeleteToken

The ContextDeleteToken is issued by either the context initiator or the target to indicate to the other party that the context is to be deleted.

```
ContextDeleteToken ::=  SEQUENCE  {
    cdtContents     [0]     CDTContents,
    cdtSeal         [1]     Seal        -- seal over cdtContents, encrypted
                            -- under the Integrity Dialogue Key
                            -- contains only the sealValue field
                        }

CDTContents ::= SEQUENCE {
    tokenType       [0]     OCTET STRING VALUE X'0301',
    SAId            [1]     OCTET STRING,
    utcTime         [2]     UTCTime OPTIONAL,
    usec            [3]     INTEGER OPTIONAL,
    seq-number      [4]     INTEGER OPTIONAL
                            }
```

cdtContents

This contains only administrative fields, identifying the token type, the context and providing exchange integrity.

seq-number

When present, this field contains a value one greater than that of the seq-number field of the last token issued from this issuer.

The other administrative fields are as described in clause 4.2.

cdtSeal

See clause 4.2 for a general description of the use of this construct.

5 Key distribution and PAC protection options

5.1 PAC protection options

[ECMA-219] specifies a framework for key distribution, but does not specify any specific key distribution schemes. It also leaves open the choice of how to protect the PAC for the distribution of privileges. The PAC choices that are relevant in this context are:

- Sealed PAC for PACs protected with symmetric cryptographic algorithms
- Signed PAC for PACs protected with asymmetric cryptographic algorithms.

In clause 7 these are combined with key distribution options to form specific mechanism options that can be specified at the GSS-API.

5.2 Key Distribution schemes

The ECMA security mechanism defines a basic set of key distribution schemes and associated data elements as described below. Note that the GetKI and ProcessKI operations identified in these descriptions are fully specified in [ECMA-219]. The TargetKeyBlock construct including its kdSchemeOID field are also specified in [ECMA-219] but are repeated in clause 5.3.1 for ease of reference.

The key distribution schemes below depend upon the existence of long term cryptographic keys which can be held by Target AEFs and KD-servers. The ECMA GSS-API allows for these keys to be either symmetric or asymmetric . Long term symmetric keys of Target AEFs are always shared between the AEF and its KD-Server. In the case where the long term keys are asymmetric we speak of the KD-server’s private key, or the Target AEF's private key.

Initiators may also possess symmetric or asymmetric keys. In the case where an initiator possesses a symmetric key this will have been established as a result of an earlier authentication, and it is shared with a KD-Server.

NOTE
Details of how authentication was performed is out of scope of this standard. However [ECMA-219] describes authentication methods and service interfaces consistent with this standard.

5.2.1 Basic symmetric key distribution scheme

For this scheme, the KD-Scheme name is : symmIntradomain

In this scheme, the initiator and the Target AEF each share different secret keys with the same KD-Server.

To establish a basic key between an initiator and a Target AEF, the initiator KD-Server returns, as a result of a GetKI operation, a targetKeyBlock containing a basic key encrypted under the Target AEF's long term secret key. On receipt of the targetKeyBlock, the Target AEF can extract the basic key directly from it.

5.2.2 Symmetric key distribution scheme with symmetric KD-Servers

For this scheme, the KD-Scheme name is : symmInterdomain

In this scheme, the initiator shares a key with a KD-Server that is different from the KD-Server with which the Target AEF shares its long term key. In addition, the KD-Servers share another long term secret key with each other.

To establish a basic key between an initiator and a Target AEF, the initiator KD-Server returns, as a result of a GetKI operation, a targetKeyBlock containing a basic key encrypted under the long term secret key shared between the two KD-Servers. On receipt of the targetKeyBlock, the Target AEF transmits it to its own KD-Server, using ProcessKI, and gets back the basic key re-encrypted under the long term secret key it shares with its KD-Server.

5.2.3 Symmetric key distribution scheme with asymmetric KD-Servers

For this scheme, KD-Scheme name is : hybridInterdomain

In this scheme, the initiator shares a key with a KD-Server that is different from the KD-Server with which the Target AEF shares its long term key. In addition, each KD-Server possesses a private/public key pair.

To establish a basic key between an initiator and a Target AEF, the initiator KD-Server returns, as a result of a GetKI operation, a targetKeyBlock containing a basic key encrypted under a temporary key and the temporary key encrypted under the Target AEF KD-Server's public key. The targetKeyBlock is also signed using the initiator KD-Server's private key.

On receipt of the targetKeyBlock, the Target AEF transmits it to its KD-Server, using ProcessKI, and gets it back with the basic key re-encrypted under the long term secret key it shares with its KD-Server.

5.2.4 Asymmetric initiator / symmetric target key distribution scheme

For this scheme, KD-Scheme name is : asymmInitToSymmTarget

In this scheme, the initiator does not use a KD-Server, but possesses a private/public key pair. The Target AEF shares a long term secret key with a KD-Server which possesses a private/public key pair.

To establish a basic key between an initiator and a Target AEF, the initiator constructs a targetKeyBlock containing a basic key encrypted with a the Target AEF KD-Server's public key. The targetKeyBlock is signed using the initiator's private key.

On receipt of the targetKeyBlock, the Target AEF transmits it to its KD-Server, using ProcessKI, and gets it back with the basic key re-encrypted under the long term secret key it shares with its KD-Server.

5.2.5 Symmetric initiator / asymmetric target key distribution scheme

For this scheme, KD-Scheme name is : symmInitToAsymmTarget

In this scheme, the Target AEF does not use a KD-Server, but possesses a private/public key pair. The initiator uses a KD-Server that has a private/public key pair, and shares a secret key with it.

To establish a basic key with a Target AEF, the initiator KD-Server returns, as a result of a GetKI operation, a targetKeyBlock containing a basic key encrypted with the Target AEF's public key. The targetKeyBlock is signed with the initiator KD-Server's private key.

On receipt of the targetKeyBlock, the Target AEF directly establishes a basic key from it.

5.2.6 Full public key distribution scheme

For this scheme, KD-Scheme name is : asymmetric

In this scheme, neither the initiator nor the Target AEF uses a KD-Server. Both the initiator and the Target AEF possesses a private/public key pair.

To establish a basic key with a Target AEF, the initiator constructs a targetKeyBlock containing a basic key encrypted under the Target AEF's public key. The targetKeyBlock is signed with the initiator's private key.

On receipt of the targetKeyBlock, the Target AEF directly establishes a basic key from it.

5.3 Key distribution data elements

In this clause we define the data elements for each of the key distribution schemes described above.

All of the key distribution schemes use the TargetKeyBlock data element defined in [ECMA-219]. The TargetKeyBlock is sent as a part of the initial context token. For clarity the TargetKeyBlock definition is repeated here. Data elements that are the same as those used in [Kerberos] are defined in annex B. Data elements that are the same as those used in [SPKM] are defined in annex C.

5.3.1 KD-Scheme independent data elements

```
TargetKeyBlock ::= SEQUENCE {
    initiatorKDSname  [0]   identifier            OPTIONAL
    kdSchemeOID       [2]   OBJECT IDENTIFIER,
    targetKDSpart     [3]   ANY                   OPTIONAL,
                                                      -- depending on kdSchemeOID
    targetPart        [4]   ANY                   OPTIONAL}
                                                      -- depending on kdSchemeOID
```

The following table shows the different syntaxes used for targetKDSpart and targetPart for the six defined KD-schemes. “Missing” in the tables means that the relevant construct is not supplied.

KD-Scheme name	kdSchemeOID	targetKDSpart	targetPart
symmIntradomain	**{kd-schemes 1}**	**Missing**	**Ticket**
symmInterdomain	**{kd-schemes 2}**	**Ticket**	**Missing**
hybridInterdomain	**{kd-schemes 3}**	**PublicTicket**	**Missing**
asymmInitToSymmTarget	**{kd-schemes 4}**	**SPKM_REQ**	**Missing**
symmInitToAsymmTarget	**{kd-schemes 5}**	**SPKM_REQ**	**Missing**
asymmetric	**{kd-schemes 6}**	**Missing**	**SPKM_REQ**

Table 1 - Key Distribution Scheme OBJECT IDENTIFIERs

The syntax of PublicTicket is given in clause 5.3.3, and the syntax of Ticket is given in annex B. The syntax of SPKM_REQ is given in annex C.

We now define the new ASN.1 OBJECT IDENTIFIERs and token construct data types required for these KD-schemes. Note that the ECMA GSS-API mechanism heavily re-uses existing data structures defined in [SPKM] and [Kerberos] as well as those defined in [ECMA-219].

5.3.2 Key distribution scheme OBJECT IDENTIFIERs

The OBJECT IDENTIFIERs that are for use in the kdSchemeOID field of TargetKeyBlock are formally derived from the kd-schemes OBJECT IDENTIFIER imported from [ECMA-219] as follows:

```
symmIntradomain          OBJECT IDENTIFIER ::= {kd-schemes 1}
symmInterdomain          OBJECT IDENTIFIER ::= {kd-schemes 2}
hybridInterdomain        OBJECT IDENTIFIER ::= {kd-schemes 3}
asymmInitToSymmTarget    OBJECT IDENTIFIER ::= {kd-schemes 4}
symmInitToAsymmTarget    OBJECT IDENTIFIER ::= {kd-schemes 5}
asymmetric               OBJECT IDENTIFIER ::= {kd-schemes 6}
```

For ease of reference, the construction of the kd-schemes OBJECT IDENTIFIER is given in annex E

The SPKM_REQ construct used in schemes 4, 5 and 6 requires a sequence of key establishment algorithm identifier values to be inserted into the key_estb_set field. The OBJECT IDENTIFIER below is defined as the (single) key establishment "algorithm" for the ECMA mechanism:

```
gss-key-estb-alg AlgorithmIdentifier ::= {kd-schemes, NULL }
```

kd-schemes

This OBJECT IDENTIFIER is the top of the arc of key distribution scheme OBJECT IDENTIFIERs defined in this Standard. It is defined in [ECMA-219].

symmIntradomain

This OBJECT IDENTIFIER indicates the basic symmetric scheme described in clause 5.2.1. As indicated in the third column of table 1, the targetKDSpart of the TargetKeyBlock is not supplied and the targetPart contains a Kerberos Ticket (see [Kerberos] and annex B). The profile of the ticket that is supported this scheme can be found in table 2.

symmInterdomain

This OBJECT IDENTIFIER indicates the symmetric interdomain scheme described in clause 5.2.2. As indicated in the fourth column of table 1, the targetPart of the TargetKeyBlock is not supplied and the targetKDSpart contains a Kerberos Ticket. The profile of the ticket that is supported in this scheme can be found in table 2.

hybridInterdomain

This OBJECT IDENTIFIER indicates the hybrid scheme described in clause 5.2.3. The targetKDSpart contains a PublicTicket (defined in clause 5.3.3). The targetPart field is not supplied. The PublicTicket contains a Kerberos Ticket. The profile supported in this scheme can be found in table 3.

asymmInitToSymmTarget

This OBJECT IDENTIFIER indicates the scheme described in clause 5.2.4. The targetKDSPart contains an SPKM_REQ (defined in annex C) whilst the targetPart is empty. The profile of SPKM_REQ that is supported in this scheme is given in table 4.

symmInitToAsymmTarget

This OBJECT IDENTIFIER indicates the scheme described clause 5.2.5. The targetPart contains an SPKM_REQ (defined in annex C) whilst the targetKDSpart is empty. The profile of SPKM_REQ that is supported in this scheme is given in table 4.

asymmetric

This OBJECT IDENTIFIER indicates the scheme described in clause 5.2.6. The targetKDSpart is not supplied and the targetPart contains an SPKM_REQ. The syntax of SPKM_REQ is given in annex C. The profile of SPKM_REQ that is supported in this scheme is given in table 4.

gss-key-estb-alg

This AlgorithmIdentifier identifies the key establishment algorithm value to be used within the key_estb_set field of an SPKM_REQ data element as the one defined by ECMA.

This algorithm is used to establish a symmetric key for use by both the initiator and the target AEF as part of the context establishment. The corresponding key_estb_req field of the SPKM_REQ will be a BIT STRING the content of which is a DER encoding of the KeyEstablishmentData element defined later.

5.3.3 Hybrid inter-domain key distribution scheme data elements

```
PublicTicket ::= SEQUENCE{
    krb5Ticket      [0]   Ticket,                    -- see annex B
    publicKeyBlock  [1]   PublicKeyBlock}
```

krb5Ticket

The Kerberos Ticket which contains the basic key. The encrypted part of this ticket is symmetrically encrypted using the key found within the encryptedPlainKey field of the KeyEstablishmentData in the PublicKeyBlock.

publicKeyBlock

Contains the key used to protect the krb5Ticket encrypted using the public key of the recipient and signed by the encryptor (i.e. the context initiator's KD-Server).

```
PublicKeyBlock ::= SEQUENCE{
    signedPKBPart[0]    SignedPKBPart,
    signature       [1]   Signature    OPTIONAL,
                              -- imported from [ECMA-219], see also annex E
    certificate     [2]   Certificate  OPTIONAL}
                              -- imported from [ISO/IEC 9594-8], see also annex E
```

signedPKBPart

The part of the publicKeyBlock which is signed.

signature

- Contains the signature calculated by the issuingKDS on the signedPKBPart field.

certificate

- If present, contains the public key certificate of the issuing KD-Server.

```
SignedPKBPart ::= SEQUENCE{
    keyEstablishmentData    [0]  KeyEstablishmentData,
    encryptionMethod        [1]  AlgorithmIdentifier        OPTIONAL,
    issuingKDS              [2]  Identifier,
    uniqueNumber            [3]  UniqueNumber,
    validityTime            [4]  TimePeriods,
                                     -- UniqueNumber and TimePeriods are
                                     -- imported from [ECMA-219], see also annex E
    creationTime            [5]  UTCTime}
```

keyEstablishmentData

Contains the KeyEstablishementData (defined in clause 5.3.4), i.e. the actual encrypted temporary key.

encryptionMethod

Indicates the algorithm used to encrypt the encryptedKey.

issuingKDS

Name of the KD-Server who produced the PublicTicket.

uniqueNumber

Value which prevents replay of the PublicTicket.

validityTime

Specifies the times for which the PublicTicket is valid.

creationTime

Contains the time at which the PublicTicket was created.

5.3.4 Key establishment data elements

The key establishment data structure represents an encrypted symmetric key along with the name of the target for which the key is encrypted. For security reasons, before encryption, the plain key is concatenated with the result of a hash function applied to the plain key itself, along with the issuer KD-Server name.

```
KeyEstablishmentData ::= SEQUENCE {
    encryptedPlainKey   [0]  BIT STRING,          -- encrypted PlainKey
    targetName          [1]  Identifier           OPTIONAL,
    nameHashingAlg      [2]  AlgorithmIdentifier  OPTIONAL}
```

encryptedPlainKey

Contains the encrypted key. The BIT STRING contains the result of encrypting a PlainKey structure.

targetName

If present, contains the name of the target application. This is necessary for some of the ECMA KD-schemes.

nameHashingAlg

Specifies the algorithm which is used to calculate the hashedName field of the PlainKey.

```
PlainKey ::= SEQUENCE {
    plainKey        [0]   BIT STRING,        -- The cleartext key
    hashedName      [1]   BIT STRING}
```

plainKey

Contains the actual bits of the plaintext key which is to be established.

hashedName

A hash of the name of the encrypting KD-Server calculated using the plainkey and KD-Server name as input (within the HashedNameInput structure). The algorithm identified in nameHashingAlg is used to calculate this value.

```
HashedNameInput ::= SEQUENCE {
    hniPlainKey     [0]   BIT STRING,        -- the same value as plainKey
    hniIssuingKDS   [1]   Identifier}
```

hniPlainKey
hniIssuingKDS

Used as input to a hashing algorithm as a general means of preventing ciphertext stealing attacks.

5.3.5 Kerberos Data elements

The full ASN.1 for the Kerberos elements used by the ECMA GSS-API mechanism is given in annex B. This clause specifies the specific contents of the Kerberos Ticket's authorization_data field required by the ECMA GSS-API mechanism.

Essentially this construct contains the PPID of the context initiator, as formally defined below.

```
ECMA-AUTHORISATION-DATA-TYPE ::= INTEGER { ECMA-ADATA (65) }

ECMA-AUTHORISATION-DATA ::= SEQUENCE {
    ecma-ad-type    [0] ENUMERATED { ppidType   (0)},
    ecma-ad-value   [1] CHOICE       { ppidValue  [0]  SecurityAttribute}}
                                -- only one choice for now
```

ppidType

Indicates the type of the ECMA authorisation data which is included in the Ticket (always zero).

ppidValue

This value is used in the ppQualification PAC protection method as defined in [ECMA-219]

5.3.6 Profiling of KD-schemes

The following tables provide profiling information for the data elements defined above and in annexes B and C. The tables indicate which optional fields must be present for each of the KD-Schemes and indicate the values which are required to be present in all fields.

5.3.6.1 Profile of Ticket (symmIntradomain and symmInterdomain)

Ticket	symmIntradomain	symmInterdomain
tkt-vno	5	
realm	ticket issuer's domain name in Kerberos realm name form	
sname	target application name including the realm of the target	
- EncTicketPart	encrypted with long term key of target AEF	encrypted with symmetric key shared between KD-Servers
-- flags	only bits 6, 10 and 11 can be meaningful in the context of the ECMA mechanism, the rest are ignored	
-- key	the basic key	
-- crealm	initiator domain name in Kerberos realm name form	
-- cname	principal name of the initiator (in the case of delegation the cname will be that of the delegate)	
-- transited	not used	
-- authtime	the time at which the initiator was authenticated	
-- starttime	not used	
-- endtime	the time at which the ticket becomes invalid	
-- renew-till	not used	
-- caddr	not used	
-- authorization-data	contains the PPID corresponding to cname	

Table 2 - Kerberos ticket fields supported

5.3.6.2 Profile of PublicTicket (hybridInterdomain)

PublicTicket	**hybridInterdomain**
krb5Ticket	
- tkt-vno	5
- realm	initiator domain name in Kerberos realm name form
- sname	target application name including the realm of the target
-- EncTicketPart	encrypted with temporary key (which is in turn encrypted within the keyEstablishmentData field)
--- flags	only bits 6, 10 and 11 can be meaningful in the context of the ECMA mechanism, the rest are ignored
--- key	the basic key
--- crealm	initiator domain name in Kerberos realm name form
--- cname	principal name of the initiator (in the case of delegation the cname will be that of the delegate)
--- transited	not used
--- authtime	the time at which the initiator was authenticated
--- starttime	not used
--- endtime	the time at which the ticket becomes invalid
--- renew-till	not used
--- caddr	not used
--- authorization-data	contains the PPID corresponding to cname
publicKeyBlock	
- signedPKBPart	
-- encryptedKey	KeyEstablishmentData structure
-- encryptionMethod	gss-estb-alg
-- issuingKDS	X.500 name of initiator's KDS (the signer)
-- uniqueNumber	creation time of publicKeyBlock plus a random bit string
-- validityTime	only one period allowed
-- creationTime	creation time of publicKeyBlock
- signature	contains all the signing information as well as the actual signature bits
- certificate	optional

Table 3 - PublicTicket fields supported

5.3.6.3 Profile of SPKM_REQ (asymmInitToSymmTarget, symmInitToAsymmTarget, asymmetric)

SPKM_REQ	asymmInitTo-SymmTarget	symmInitTo-AsymmTarget	asymmetric
requestToken			
- tok_id	not used - fixed value of '0'		
- context_id	not used - fixed value of bit string containing one zero bit		
- pvno	not used - fixed value of bit string containing one zero bit		
- timestamp	creation time of SPKM_REQ - required		
- randSrc	random bit string		
- targ_name	X.500 Name of KD-Server of target	X.500 Name of target AEF	X.500 Name of target AEF
- src_name	X.500 Name of initiator	X.500 Name of initiator's KD-Server	X.500 Name of initiator
- req_data			
-- channelId	not used - octet string of length one value '00'H		
-- seq_number	missing		
-- options	not used - all bits set to zero		
-- conf_alg	not used - use NULL CHOICE		
-- intg_alg	not used - use a SEQUENCE OF with zero elements		
- validity	mandatory		
- key_estb_set	only one element supplied containing ecma-gss-key-estb-alg		
- key_estb_req	contains KeyEstablishment-Data with targetApplication field supplied	contains KeyEstablishmentData with targetApplication field missing	
- key_src_bind	missing		
req_integrity	sig_integ mandatory		
certif_data	only userCertificate field supported	only userCertificate field supported	only userCertificate field supported
auth_data	missing	missing	missing

Table 4 - SPKM_REQ fields supported

5.4 Returned Key Scheme Information

This clause defines the helpful information that may be returned from an acceptor for each specific ECMA key distribution scheme. The information is expected to be sent from the acceptor to the initiator during mechanism negotiation in order to support the scheme used by the mechanism option being selected.

The information is returned in the form of a SEQUENCE OF Directory attributes, so that in cases where the information is missing, it can potentially be retrieved from a Directory.

The specific attribute types returned depends on the key distribution scheme that will be used between the actual context initiator and acceptor concerned. The attribute types that are returned for each of the six key distribution schemes are shown below:

SCHEME	**SYNTAX**	**SEMANTICS**
symmIntradomainInfo	NULL	no information returned
symmInterdomainInfo	NULL	no information returned
hybridInterdomainInfo	SEQUENCE OF Certificate	containing target's KD-Server Certificate and related Certificates
asymmInitToSymmTargetInfo	SEQUENCE OF Certificate	containing target's KD-Server Certificate and related Certificates
symmInitToAsymmTargetInfo	SEQUENCE OF Certificate	containing target AEF's Certificate and related Certificates
asymmetricInfo	SEQUENCE OF Certificate	containing target AEF's Certificate and related Certificates

6 Algorithm use within ECMA mechanism

Cryptographic and hashing algorithms are used for various purposes within the ECMA GSS-API mechanism. This clause categorises these algorithms according to usage so that context initiators and acceptors can more easily determine if they have the cryptographic support required to allow inter-operation. The categorisation is then refined into cryptographic profiles that can be incorporated into specific mechanism identifiers for the purpose of mechanism negotiation. This is done in clause 7.

Table 5 summarises the different uses to which algorithms are put within the ECMA GSS-API mechanism.

Use Reference	Description of use	Type of Algorithm
1	PAC protection using seal	OWF + symmetric integrity
2	PAC protection using signature	OWF + asymmetric signature
3	basic key usage	symmetric confidentiality and integrity
4	integrity dialogue key derivation	OWF
5	integrity dialogue key usage	symmetric integrity
6	CA public keys	OWF + asymmetric signature
7	encryption of security data using shared long term symmetric key	symmetric confidentiality.
8	name hash to prevent ciphertext stealing	OWF
9	asymmetric basic key distribution	asymmetric encryption and OWF + signature
10	ECMA key estab. within SPKM_REQ	(fixed value for ECMA)
11	confidentiality dialogue key derivation	OWF
12	confidentiality dialogue key use (on user data)	symmetric confidentiality

Table 5 - Summary of algorithm uses:

The algorithms can now be further categorised into broader classes as follows:
Class 1: symmetric for security of mechanism: Uses 1, 3, 5, 7
Class 2: all OWFs: Uses 1, 2, 4, 6, 8, 11
Class 3: internal mechanism asymmetric, encrypting: Use 9
Class 4: internal mechanism asymmetric, non-encrypting: Use 2
Class 5: CA's asymmetric non-encrypting: Use 6
Class 6: User data confidentiality, symmetric: Use 12

Use 10 is a fixed value for ECMA, and does not contribute to mechanism use options. The fixed value for this has already been defined in clause 5.3.2.

Based on these classes, the following cryptographic algorithm usage profiles are defined. Other profiles are possible and can be defined as required. Note that symmetric algorithm key sizes are included in this profiling, thus DES/64 indicates DES with a 64 bit key.

	Profile 1: Full	Profile 2: No user data Confidentiality	Profile 3: Exportable	Profile 4: No Asymm Encryption	Profile 5: Defaulted
Class 1	DES/64	DES/64	RC4/128	DES/64	separately agreed default
Class 2	MD5	MD5	MD5	MD5	separately agreed default
Class 3	RSA	RSA	RSA	Not supported	separately agreed default
Classes 4 and 5	RSA	RSA	RSA	DSS/***	separately agreed default
Class 6	DES/64	None	RC4/40	DES/64	separately agreed default

Table 6 - Algorithm profiles

Where:

- Profile 1 provides full security, using standard cryptographic algorithms with commonly accepted key sizes.
- Profile 2 is the same but without supporting any confidentiality of user data.
- Profile 3 is designed to be exportable under many countries' legislations,
- Profile 4 provides security at the same level of quality as Profile 1 but uses the Digital Signature Standard instead of RSA. In this scheme only symmetric key distribution is possible, of the key distribution methods described in clause 5.
- Profile 5 uses algorithms identified by a separately specified default. It is intended for use by organisations who wish to use their own proprietary or government algorithms by separate agreement or negotiation.

Clause 7 shows how these algorithm profiles can be used to extend the architectural key distribution schemes to form negotiable ECMA mechanism choices.

7 Identifiers for ECMA mechanism choices

Preceding clauses have separately defined the alternatives allowed by the generic ECMA mechanism in terms of PAC protection, key distribution schemes and the use of cryptographic and hash algorithms within the data elements.

This clause brings these together by defining the specific ECMA mechanism identifiers which correspond to each combination of the available options under these headings. These specific mechanism identifiers are intended to be negotiable using a generic GSS-API negotiation scheme.

7.1 Architectural mechanism identifiers

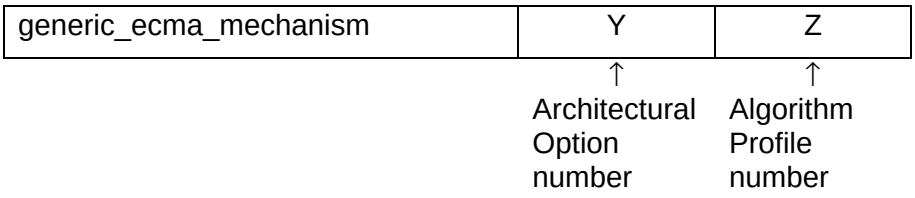
The approach is first to combine the PAC protection options (sealed or signed) in a simple way with key distribution schemes to form broad architectural mechanism options, as follows:

Architectural Mechanism Number	Description of Architecture Option	PAC prot'n	Key Distribution Scheme(s)
1	Full Symmetric Cryptography	Sealed	symmIntradomain; symmInterdomain
2	Symmetric key distribution	Signed	symmIntradomain; symmInterdomain
3	Symmetric initiator and target; Asymmetric KD-Servers;	Signed	symmIntradomain; hybridInterdomain
4	Asymmetric initiator ; Symmetric target; Asymmetric KD-Server for target;	Signed	asymmInitTo SymmTarget
5	Symmetric Initiator with Asymmetric KD-Server, and Asymmetric Target	Signed	symmInitTo AsymmTarget
6	Asymmetric Initiator and Target	Signed	asymmetric

Table 7 - ECMA Architecture Mechanism Options

Each of the ECMA architecture mechanism options described above represents a combination of one of these PAC protection options and a key distribution scheme. The six key distribution schemes are defined in clause 5.2.

Generic GSS-API mechanism negotiation will be carried out on the basis of the generic ECMA mechanism OBJECT IDENTIFIER concatenated with an architectural mechanism number from table 7, and an algorithm profile reference number from clause 6. Thus the form of a negotiable ECMA mechanism is:



Thus an ECMA mechanism using a fully symmetric key distribution scheme and an exportable cryptographic algorithm profile would have an OBJECT IDENTIFIER of:

{ generic_ecma_mech (2) (3) }

An ECMA mechanism using a fully asymmetric initiator and target architectural scheme, and an algorithm profile not supporting user data confidentiality would have an OBJECT IDENTIFIER of:

{ generic-ecma-mech (6) (2) }

Not all combinations of key distribution scheme and algorithm profile are meaningful, but meaningful ones are intended to be negotiable using a generic GSS-API negotiation scheme such as [SNEGO].

8 Errors

The errors defined here are the minor_status codes that can be returned from the ECMA mechanism. Some values are newly defined in this standard, but the majority have their origin in [ECMA-219]. For the new codes, the mapping to the errors defined in [ECMA-219] is also given.

8.1 Minor Status Codes

Different implementations of the ECMA Mechanism can return different minor_status representations for the same error. This section defines (recommends) common symbolic names for the minor status codes identified for the ECMA Mechanism without assigning an actual representation to the values at the API level. These definitions enable independent implementors to enhance portability across different implementations of the ECMA Mechanism (as they do not influence the “logic” behaviour of the caller). The actual conversion of minor_status indicators to text representations is, as normal, done using gss_display_status(). Any implementation must be capable of mapping these symbolic names to and from the actual values (INTEGER or other) used to represent the minor_status codes specified here.

In order to give the minor_status code semantics as precisely as possible, annex D gives the mapping between minor_status codes and internal errors from the ECMA Mechanism’s arguments and results where they apply.

8.1.1 Non ECMA-specific codes

GSS related codes:

GSS_ECMA_S_G_VALIDATE_FAILED

Validation error

GSS_ECMA_S_G_BUFFER_ALLOC

Couldn’t allocate gss_buffer_t data

GSS_ECMA_S_G_BAD_MSG_CTX

Message Context Invalid

GSS_ECMA_S_G_WRONG_SIZE

Buffer is wrong size

GSS_ECMA_S_G_BAD_USAGE

Credential usage type is unknown

GSS_ECMA_S_G_UNAVAIL_QOP

Unavailable quality of protection specified

Implementation related codes:

GSS_ECMA_S_G_MEMORY_ALLOC

Couldn't perform requested memory allocation

8.1.2 ECMA-specific codes

Minor_status codes resulting from errors specified in this ECMA Standard:

GSS_ECMA_S_SG_SA_INCOMPLETE

Attempt to use incomplete Security Association

GSS_ECMA_S_SG_INVALID_TOKEN_DATA

Data is improperly formatted: cannot encode into token

GSS_ECMA_S_SG_INVALID_TOKEN_FORMAT

Received token is improperly formatted: cannot decode

GSS_ECMA_S_SG_SA_DELETED

Security Association deleted at peer's request

GSS_ECMA_S_SG_BAD_DELETE_TOKEN_RECD

Invalid delete token received: context not deleted

GSS_ECMA_S_SG_INVALID_SAID

The SAId between the initiator and acceptor is already in use

GSS_ECMA_S_SG_INVALID_TARGET_AEF_PROT

The validation of the Seal on the Target AEF part of the token failed

GSS_ECMA_S_SG_TOKEN_TIME_NOT_YET_VALID

The time given in the Token has not yet been reached by the acceptor

GSS_ECMA_S_SG_TOKEN_TOO_OLD

The time given in the Token is too old for the acceptor

GSS_ECMA_S_SG_BAD_CONTEXT_FLAGS

The context flags are not supported by the acceptor

GSS_ECMA_S_SG_INVALID_CHANNEL_BINDINGS

The channel bindings in the token are not valid

GSS_ECMA_S_SG_BAD_KD_SCHEME

The KD scheme in the target key block is not supported by the acceptor

GSS_ECMA_S_SG_INVALID_TARGET_ID

The target ID in the targetAEFpart in the token is not known by the receiving AEF

Minor_status codes resulting from error specified in [ECMA-219]:

GSS_ECMA_S_SG_SERVER_SA_ALREADY_ESTABLISHED

Security Association to be opened with a security server is already open

GSS_ECMA_S_SG_INCOMP_CERT_SYNTAX

Invalid Certificate: incompatible syntax version of cert, or cert specific, contents

GSS_ECMA_S_SG_BAD_CERT_ATTRIBUTES

Invalid Certificate Specific contents: unacceptable security attributes for authentication, access control or protection

GSS_ECMA_S_SG_INVAL_TIME_FOR_ATTRIB

Invalid Certificate Specific contents: current time outside specified timeperiods in PAC

GSS_ECMA_S_SG_PAC_RESTRICTIONS_PROB

Invalid Certificate Specific contents: invalid mandatory restriction in PAC

GSS_ECMA_S_SG_ISSUER_PROBLEM

Invalid Certificate: not issued by a trusted authority

GSS_ECMA_S_SG_CERT_TIME_TOO_EARLY

Invalid Certificate: not yet reached validity period

GSS_ECMA_S_SG_CERT_TIME_EXPIRED

Invalid Certificate: validity period expired

GSS_ECMA_S_SG_INVALID_CERT_PROT

Invalid Certificate: invalid or wrong protection mechanism

GSS_ECMA_S_SG_REVOKED_CERT

Invalid Certificate: the certificate has been revoked

GSS_ECMA_S_SG_KEY_CONSTR_NOT_SUPP

No requested key construction data types supported by the KD-Server

GSS_ECMA_S_SG_INIT_KD_SERVER_UNKNOWN

Unknown initiator KD-Server at the target or target KD-Server

GSS_ECMA_S_SG_INIT_UNKNOWN

Unknown initiator at the target or target KD-Server

GSS_ECMA_S_SG_INSUFF_AUTHORISATION

Insufficient authorisation: access control failure for operation

GSS_ECMA_S_SG_ALG_PROBLEM_IN_DIALOGUE_KEY_BLOCK

Invalid Dialogue Key Block: Algorithm(s) not supported

GSS_ECMA_S_SG_NO_BASIC_KEY_FOR_DIALOGUE_KEY_BLOCK

Invalid Dialogue Key Block: no Basic Key to derive Dialogue Key Block from

GSS_ECMA_S_SG_KEY_DISTRIB_PROB

Invalid target key block: bad KD scheme or Key Info for KD-Server or target”

GSS_ECMA_S_SG_INVALID_USER_CERT_IN_KEY_BLOCK

Invalid Key Block: invalid user certificate

GSS_ECMA_S_SG_OPERATION_NOT_SUPP

Operation not supported by initiator or target Security Server

GSS_ECMA_S_SG_SEC_ASSOC_ID_FAILURE

Unknown Security Association identifier specified for Security Server or target

GSS_ECMA_S_SG_UNACCEPTABLE_ACT_REQ

Requirements specified for an ACT are unacceptable to PA-Server

GSS_ECMA_S_SG_UNSPECIFIED

Unknown or not released reason for the error

8.2 Quality of protection

The specification of quality of protection on a per message basis is not supported by the ECMA mechanism. Quality of protection is determined by the contents of the DialogueKeyBlock in the TargetAEFPart of the InitialContextToken.

9 Support functions

So that its users can gain full benefit from its facilities, the ECMA mechanism requires support functions to make PAC attributes available to the GSS-API application (attribute handling support functions), and to set and get control information relating to the validity and type of acceptor of a security context (acceptor control and support functions).

The functions required are defined below.

9.1 Attribute handling support functions

Three attribute handling support functions are defined :

GSS_Set_cred_attributes: To enable a GSS-API context initiator to specify PAC privilege and miscellaneous attributes to be included in the caller credentials in order to become part of a security context. Miscellaneous attributes include issuer domain name and validity time periods.

GSS_Get_sec_attributes: To extract PAC information, either from a GSS-API context, or from a credential handle. Both Privilege Attributes and miscellaneous attributes can be retrieved from the PAC. The function can be invoked either by a context initiator or by a context acceptor.

GSS_Get_received_creds: To extract credential handles from a GSS-API context (established with GSS_Accept_sec_context function). This is only applicable when traced or composite delegation is involved (see clause 9.2 for a definition of these forms), and can only be invoked by a context acceptor. A call to GSS_Get_received_creds is an intermediary step for an acceptor, before extracting each delegate's PAC information with a call to GSS_Get_sec_attributes.

A security attribute at the extended GSS-API interface is defined as:

```
SecAttribute ::= {
    attributeType       OBJECT IDENTIFIER,
    definingAuthority   OCTET STRING          OPTIONAL,
    securityValue       OCTET STRING }
```

NOTE
This is a simplified form of the full SecurityAttribute syntax defined in [ECMA-219], and repeated in annex E.

attributeType

Defines the type of the attribute. Attributes of the same type have the same semantics.

definingAuthority

The authority responsible for the definition of the semantic of the value of the security attribute.

securityValue

The value of the security attribute. Its syntax is determined by the attribute type.

9.1.1 GSS_Set_cred_attributes

Input :

- cred_handle OCTET STRING,
- required_attributes SET OF SecAttribute,
- new_cred_req BOOLEAN
- commit_cred_req BOOLEAN

Output :

- output_cred_handle OCTET STRING

Return major_status code:

- GSS_S_COMPLETE indicates that the nominated attributes are permitted to the caller and have been set.
- GSS_S_CREDENTIALS_EXPIRED indicates that the specified credentials have expired.
- GSS_S_DEFECTIVE_CREDENTIAL indicates that defective credentials have been detected.
- GSS_S_FAILURE indicates a failure, unspecified at the GSS-API level.
- GSS_S_UNAUTHORIZED indicates that the function, or an argument of the function was not authorised.
- GSS_S_UNAVAILABLE indicates that the operation is not supported.

This function enables a caller to request a set of privileges and miscellaneous attributes, optionally replacing existing credentials or creating a new set. The effect of this interface is not cumulative, the requested attributes replace any existing attributes in the credentials claimed.

Parameters for GSS_Set_cred_attributes:

cred_handle

Handle for credentials claimed, cred_handle refers to an authenticated principal. Supply NULL to use default credentials.

required_attributes

A set of required privilege and miscellaneous attributes. NULL specifies default attributes to be requested. Otherwise, only the privilege and miscellaneous attributes specified will be present.

If a specified attribute is provided with a NULL value field, the value allocated to the attribute will be the default for the specified attribute available to the authenticated principal according to the prevailing security policy. Otherwise the value specified will be that present. If a value specified clashes with policy, an error is returned.

If a role name is specified as a single attribute required, and policy permits the principal to use it, it will be used as an attribute set reference to select a set of attributes and acceptor controls according to policy.

If a role name is specified along with other required attributes, and policy permits the principal to use the role name, the attributes potentially available for the authenticated principal are taken from a set compounded of the principal's authorised attributes, and the attributes associated with the role name.

new_cred_req

TRUE for a new credentials set, FALSE replaces the original.

commit_cred_req

TRUE for immediate attribute acquisition, FALSE for deferred attribute acquisition.

output_cred_handle

The credentials handle for the changed or new credentials.

GSS_Set_cred_attributes produces a modified version of the input credentials (cred_handle). The original credentials are changed if new_cred_req is FALSE, otherwise the output_cred_handle references a new, and different, copy of the original input credentials (which remain untouched). GSS_Release_cred can be used when the caller is finished with any new credentials created by this function.

9.1.2 GSS_Get_sec_attributes

Input :

- cred_handle OCTET STRING,
- context_handle INTEGER,
- attribute_types_required SET OF OBJECT IDENTIFIER

Output :

- priv_attributes SET OF SecAttribute
- misc_attributes SET OF SecAttribute

Return major_status code :

- GSS_S_COMPLETE indicates that retrieval of attributes is supported and that all, some, or none of the requested attribute types have been returned.
- GSS_S_CONTEXT_EXPIRED indicates that the specified security context has expired.
- GSS_S_CREDENTIALS_EXPIRED indicates that the specified credentials have expired.
- GSS_S_DEFECTIVE_CREDENTIAL indicates that defective credentials have been detected.
- GSS_S_FAILURE indicates a failure, unspecified at the GSS-API level.
- GSS_S_UNAVAILABLE indicates that the operation is not supported.

This function can be used by context initiators and context acceptors to query attributes in credentials or security contexts. If the credentials or security context represents a delegation chain and contains multiple PACs, attributes are retrieved only from the first of them. If the attribute_types_required parameter is not supplied, then all attribute types from the PAC are returned. This option could allow clients of this interface to query all attributes and pass Privilege Attributes to a separate authorisation service to make a decision. To obtain PAC attributes from intermediate PACs in a delegation chain, the caller should first call GSS_Get_delegate_creds (see clause 9.1.3).

Parameters for GSS_Get_sec_attributes:

cred_handle

Handle to credentials, cred_handle refers to an authenticated principal. Supply NULL to use default credentials, or a context handle. Note that NULL, without a context handle, is only used for obtaining the caller's own attributes.

context_handle

GSS-API security context handle, context_handle refers to a context that is part of an established Security Association. Context_handle is ignored if a non-NULL cred_handle is presented. (Note: it is typically only necessary to use a context_handle parameter rather than cred_handle for the case when a security context is emitted by gss_accept_sec_context, but not with an accompanying set of delegated credentials).

attribute_types_required

A set of security attribute types. If the default (NULL) is specified, then all miscellaneous and Privilege Attributes are returned.

This standard does not specify which attributes must be supported, but annex B defines some common security attributes that are appropriate.

priv_attributes

A set of Privilege Attributes. Response is conditional on the "attribute_types_required" input.

misc_attributes

A set of miscellaneous attributes. Response is conditional on the "attribute_types_required" input.

9.1.3 GSS_Get_received_creds

Input :

– context_handle INTEGER,

Output :

– received_creds SEQUENCE OF OCTET STRING

Return major_status code :

– GSS_S_COMPLETE	indicates that the requested delegate credentials were retrieved.
– GSS_S_CONTEXT_EXPIRED	indicates that the specified security context has expired.
– GSS_S_FAILURE	indicates a failure, unspecified at the GSS-API level.
– GSS_S_UNAUTHORIZED	indicates that the function, or an argument of the function was not authorised.
– GSS_S_UNAVAILABLE	indicates that the operation is not supported.

This function supports the retrieval of all credentials received by an acceptor. It is intended for context acceptors that require not only the initiator's credentials, but also delegates' credentials, to apply their local security policy. A typical example is the retrieval of delegate credentials to subsequently obtain delegate Privilege Attributes (using GSS_Get_sec_attributes) for use in authorisation decisions.

Parameters for GSS_Get_received_creds:

context_handle

GSS-API security context handle, context_handle refers to a security context that is part of an established association. A default context is assumed if no context_handle is supplied..

received-creds

Contains an ordered list of credentials for the original initiator and for each of the intermediate delegates (if any) between the original initiator and this context acceptor, the first of these being the credentials of the original initiator, and the last being of the immediately preceding delegate. It is expected that the normal use for such credentials would merely be inspection via GSS_Get_sec_attributes as most known mechanisms would not permit such delegate credentials to be directly used for initiating further security contexts. Note that it is the caller's responsibility to free any received credentials returned from gss_get_received_creds via gss_release_cred.

9.2 Control and support functions for context acceptors

These functions enable a GSS-API context initiator to impose constraints on the security context to be established via GSS_Init_sec_context function, and enable a GSS-API context acceptor to retrieve the control information that applies to a security context established using the GSS_Accept_sec_context function, and build credentials from others..

Three support functions for context acceptor control are defined:

GSS_Set_cred_controls function: To enable a GSS-API context initiator to specify delegate/target controls to be included in the caller's credentials in order to be part of a security context. The controls determine the context acceptors with which valid Security Associations can be established using the associated credentials, and whether they can act only as delegates, only as targets or as delegate/targets. Restrictions over the operations that are authorised under the context can also be specified.

GSS_Get_sec_controls function: To enable a GSS-API context initiator or a GSS-API context acceptor to extract acceptor control information either from a credential handle or from a security context.

GSS_Compound_creds function: To enable a delegate (which is acting as a GSS-API acceptor for a context initiator, and as a GSS-API context initiator for another acceptor) to build new credentials made from the received credentials and its own credentials.

```
AcceptorControl ::= SEQUENCE {
    targetOnly          SEQUENCE OF SecAttribute OPTIONAL,
    delegateOnly        SEQUENCE OF SecAttribute OPTIONAL,
    delegateTarget      SEQUENCE OF SecAttribute OPTIONAL,
    delegationMode      DelegationMode    OPTIONAL,
                            }

DelegationMode  :=    ENUMERATED {
                            default         (0),
                            simple          (1),
                            traced          (2),
                            composite       (3)       }
```

targetOnly
delegateOnly
DelegateTarget

Specifies one or several qualifier attributes describing the targets, delegates or delegate/targets for which controls are to apply.

- the targetOnly specifies that the qualifier(s) are identifying one or more targets, none of which may use the credentials as a delegate.
- the delegateOnly choice specifies that the qualifier(s) are identifying one or more delegates, none of which should use the PAC Privilege Attributes in the credentials when authorising access to their own protected resources, but which may use the received credentials as a delegate.

NOTE
Only the acceptor system's AEF can prevent an acceptor permitting access based on attributes not intended for it. However it is not in the interests of an acceptor or its AEF to permit access to resources under their control on the basis of attributes that are explicitly stated as not being appropriate.

- the delegateTarget choice specifies that the qualifier is identifying one or more delegate/targets any of which can use the received credentials as a delegate and can also use the PAC Privilege Attributes in the credentials when authorising access to its own protected resources.

delegationMode

Indicates the mode of delegation required. A full description of delegation modes is given in [ECMA-219].

Currently three delegation modes and one default are specified:

- default: whatever mode of delegation has been set as default (this may be no delegation) is required.
- simple: only the original initiator's credentials are presented to an acceptor. Credentials of intermediate delegates are not used..
- traced: the credentials of the original initiator and of all the intermediate delegates are presented to an acceptor.
- composite: only the credentials of the original initiator and the credentials of the immediate caller are presented to an acceptor.
- exotic: the credentials to be delegated are randomly selected according to the moon position and the temperature of the room.

9.2.1 GSS_Set_cred_controls call

Input :

- cred_handle OCTET STRING,
- required_acceptor_control AcceptorControl,
- replace_old_controls BOOLEAN
- new_cred_req BOOLEAN
- commit_cred_req BOOLEAN

Output :

- output_cred_handle OCTET STRING

Return major_status code:

- GSS_S_COMPLETE indicates that the controls have been set.
- GSS_S_CREDENTIALS_EXPIRED indicates that the specified credentials have expired.
- GSS_S_DEFECTIVE_CREDENTIAL indicates that defective credentials have been detected.
- GSS_S_FAILURE indicates a failure, unspecified at the GSS-API level.
- GSS_S_UNAUTHORIZED indicates that the function, or an argument of the function was not authorised.
- GSS_S_UNAVAILABLE indicates that the operation is not supported.

This function supports requests to set context acceptor controls, optionally replacing existing credentials controls or creating a new set of credentials with new controls. The effect of this interface is either cumulative or not depending on the value of the replace_old_controls parameter.

Parameters for GSS_Set_cred_controls:

cred_handle

Handle for credentials claimed, it refers to an authenticated principal. Supply NULL to use default credentials.

required_acceptor_control

The control settings required.

replace_old_controls

TRUE to replace acceptor controls existing in original credentials. FALSE to specify additional controls.

new_cred_req

TRUE for a new credentials set, FALSE to modify the original.

commit_cred_req

TRUE for immediate action, FALSE for deferred action.

output_cred_handle

GSS_Set_cred_controls produces a modified version of the input credentials (cred_handle). The original credentials are directly changed if duplicate_cred_req is FALSE, otherwise the output_cred_handle references a new, and potentially different, copy of the original input credentials (which remain untouched). gss_release_cred can be used when the caller is finished with any new credentials created by this function.

9.2.2 GSS_Get_sec_controls

Input :

- cred_handle OCTET STRING,
- context_handle INTEGER,

Output :

– acceptor_controls SET OF AcceptorControl,

Return major_status code :

– GSS_S_COMPLETE indicates that the acceptor control information has been returned

– GSS_S_CREDENTIALS_EXPIRED indicates that the specified credentials have expired.

– GSS_S_DEFECTIVE_CREDENTIAL indicates that defective credentials have been detected.

– GSS_S_FAILURE indicates a failure, unspecified at the GSS-API level.

– GSS_S_UNAVAILABLE indicates that the operation is not supported.

This function enables a caller to enquire the current value of the acceptor controls in the specified credentials or context.

This function can be used by context initiators and context acceptors to query acceptor controls in credentials or security contexts..

Parameters for GSS_Get_sec_controls:

cred_handle

Handle to credentials. It refers to an authenticated principal. Supply NULL to use default credentials, or a context handle.

context_handle

GSS-API security context handle, context_handle refers to a security context that is part of an established association. Context_handle is ignored if a non-NULL cred_handle is presented. (Note: it is typically only necessary to use a context_handle parameter rather than cred_handle for the case when a security context is emitted by gss_accept_sec_context, but not with an accompanying set of delegated credentials).

acceptor_controls

A set of acceptor controls. Acceptor controls are described in clause 9.2.1.

9.2.3 GSS_Compound_creds call

Input :

– delegated_cred_handle OCTET STRING

– cred_handle OCTET STRING,

Output :

– cred_handle_new OCTET STRING

Return major_status code :

– GSS_S_COMPLETE indicates that the credentials were successfully compounded

– GSS_S_CREDENTIALS_EXPIRED indicates that one or more of the specified credentials have expired.

– GSS_S_DEFECTIVE_CREDENTIAL indicates that defective credentials have been detected.

– GSS_S_FAILURE indicates a failure, unspecified at the GSS-API level.

– GSS_S_UNAVAILABLE indicates that the operation is not supported.

Parameters for gss_compound_cred:

delegated_cred_handle

A handle to the credentials being delegated, it refers to one or several authenticated principals.

cred_handle

A handle to claimed credentials of the caller, cred_handle refers to an authenticated principal.

cred_handle_new

A handle to the compounded set of credentials.

9.3 Attribute specifications

For attributes that appear in the PAC with the syntax SecurityAttribute, the syntaxes of their securityValue fields are defined below.

9.3.1 Privilege attributes

Privileges are defined under the OBJECT IDENTIFIER:

privilege-attribute OBJECT IDENTIFIER ::= { iso(1) identified-organisation(3) icd-ecma(012) technical-report(1) security-in-open-systems(046) privilege-attribute(4) }

9.3.1.1 Access Identity

The access identity represents the principal's identity to be used for access control purposes.

The type of this attribute is { privilege-attribute 2 }

Its syntax in the PAC is SecurityAttribute, with a securityValue syntax of Identifier.

When it is returned by a GSS_Get_sec_attributes call, or set by a GSS_Set_cred_attributes call, the security_value field of the gss_sec_attr structure in gss_sec_attr_set contains a pointer to the gss_id structure (defined in clause 9.4)

9.3.1.2 Group

A group represents a characteristic common to several principals. A security context may contain more than one group for a given principal. The group attribute may therefore contain more than one group.

The type of this attribute is { privilege-attribute 4 }

Its syntax in the PAC is SecurityAttribute, with a securityValue syntax of SEQUENCE OF Identifier.

When it is returned by a GSS_Get_sec_attributes call, or set by a GSS_Set_cred_attributes call, the security_value field of the gss_sec_attr structure in gss_sec_attr_set contains a pointer to the gss_id_set structure (defined in clause 9.4)

9.3.1.3 Primary group

The primary group represents a unique group to which a principal belongs. A security context must not contain more than one primary group for a given principal.

The type of this attribute is { privilege-attribute 3 }

Its syntax in the PAC is SecurityAttribute, with a securityValue syntax of Identifier.

When it is returned by a GSS_Get_sec_attributes call, or set by a GSS_Set_cred_attributes call, the security_value field of the gss_sec_attr structure in gss_sec_attr_set contains a pointer to the gss_id structure (defined in clause 9.4.1)

9.3.1.4 Role attribute

The role attribute represents the principal's role. There may be a one to one mapping between a role name and a role attribute.

The type of this attribute is { privilege-attribute 1 }

Its syntax in the PAC is SecurityAttribute, with a securityValue syntax of Identifier.

When it is returned by a GSS_Get_sec_attributes call, or set by a GSS_Set_cred_attributes call, the security_value field of the gss_sec_attr structure in gss_sec_attr_set contains a pointer to the gss_id structure (defined in clause 9.4)

9.3.2 Attribute set reference

9.3.2.1 Role name

The role name is an attribute set reference used only as input parameter to a GSS_Set_cred_attributes call to select a set of security attributes for credentials. It does not appear in the PAC.

The type of this attribute is { privilege-attribute 17 }

For this attribute, the security_value field of the gss_sec_attr structure in gss_sec_attr_set contains a pointer to the gss_id structure (defined in clause 9.4)

9.3.3 Miscellaneous attributes

Miscellaneous attributes are defined under the OBJECT IDENTIFIER:

misc-attribute OBJECT IDENTIFIER ::= { iso(1) identified-organisation(3) icd-ecma(012) technical-report(1) security-in-open-systems(046) misc-attribute(3) }

9.3.3.1 Audit Identity

The access identity represents the principal's identity to be used for audit purposes.

The type of this attribute is { misc-attribute 2 }

Its syntax in the PAC is SecurityAttribute, with a securityValue syntax of Identifier.

When it is returned by a GSS_Get_sec_attributes call, or set by a GSS_Set_cred_attributes call, the security_value field of the gss_sec_attr structure in gss_sec_attr_set contains a pointer to the gss_id structure (defined in clause 9.4)

9.3.3.2 Issuer domain name

The issuer domain name represents the name of the domain which issued the principal PAC. It is carried in the PAC in the issuerDomain field. It cannot be set by a call to GSS_Set_cred_attributes.

The type of this attribute is { misc-attribute 10 }

When it is returned by a GSS_Get_sec_attributes call, the security_value field of the gss_sec_attr structure in gss_sec_attr_set contains a pointer to the gss_id structure (defined in clause 9.4)

9.3.3.3 Validity periods

The validity periods represent the time periods within which the principal PAC is valid. It is carried in the PAC in the validity and timePeriods fields.

The type of this attribute is { misc-attribute 11 }

When it is returned by a GSS_Get_sec_attributes call, or set by a GSS_Set_cred_attributes call, the security_value field of the gss_sec_attr structure in gss_sec_attr_set contains a pointer to the gss_period_list structure (defined in clause 9.4).

9.3.3.4 Optional restrictions

The optional restrictions represent restrictions that apply to the security context. The context may be accepted, even if the application is unable to understand the optional restrictions. Optional restrictions are carried in the PAC in the restrictions field.

The type of this attribute is { misc-attribute 12 }

When it is returned by a GSS_Get_sec_attributes call, or set by a GSS_Set_cred_attributes call, the security_value field of the gss_sec_attr structure in gss_sec_attr_set contains a pointer to the gss_id structure (defined in clause 9.4).

9.3.3.5 Mandatory restrictions

The mandatory restrictions represent restrictions that apply to the security context. The context must not be accepted if the application is unable to understand the mandatory restrictions. Optional restrictions are carried in the PAC in the restrictions field.

The type of this attribute is { misc-attribute 13 }

When it is returned by a GSS_Get_sec_attributes call, or set by a GSS_Set_cred_attributes call, the security_value field of the gss_sec_attr structure in gss_sec_attr_set contains a pointer to the gss_id structure (defined in clause 9.4).

9.3.4 Qualifier attributes

Qualifier attributes are defined under the OBJECT IDENTIFIER:

```
qualifier-attribute    OBJECT IDENTIFIER ::= { iso(1) identified-organisation(3) icd-ecma(012) technical-
report(1) security-in-open-systems(046) qualifier-attribute(5) }
```

9.3.4.1 Acceptor name

An acceptor name represents the name of an application that can potentially accept the security context either as a target only, or a delegate target, or delegate only. Acceptor names are carried in the PAC as protection methods. More than one name can be present.

The type of this attribute is { qualifier-attribute 1 }

When it is returned by a GSS_Get_sec_controls call, or set by a GSS_Set_cred_controls call, the security_value field of the gss_sec_attr structure in gss_control_set contains a pointer to the gss_id_set structure (defined in clause 9.4).

9.3.4.2 Application trust group

An application trust group represents a group of acceptors defined by the security administrator that mutually trust each other not to spoof each others' identity. An application trust group can potentially accept the security context either as a target only, or a delegate target, or delegate only. Application trust groups are carried in the PAC as protection methods. More than one may be present.

The type of this attribute is { qualifier-attribute 2 }

When it is returned by a GSS_Get_sec_controls call, or set by a GSS_Set_cred_controls call, the security_value field of the gss_sec_attr structure in gss_control_set contains a pointer to the gss_id_set structure (defined in clause 9.4)

9.4 C Bindings

This section specifies C language bindings for the GSS-API ECMA mechanism support functions.

9.4.1 Data types and calling conventions

The following data types :

- OM_uint32,
- gss_buffer_t,
- gss_OID,
- gss_OID_set,
- gss_cred_id_t,
- gss_ctx_id_t,

are defined in [GSS-API], along with the calling conventions.

9.4.1.1 Identifier

Identifiers have the following data structure:

```
typedef struct    {
    gss_type_en    id_type
    gss_value      id_value;
    }  gss_id;
```

Where id_type identifies the syntax within the Identifier type:

```
typedef enum    {
    gss_oid_t,          /* for OID */
    gss_integer,        /* for Integer */
    gss_string,         /* for character string */
    gss_uuid,           /* for DCE UUID */
    gss_buffer_t;       /* for gss_buffer */
    }  gss_type_en;
```

And where id_value is the actual value of the data of type Identifier:

```
struct union      {
    gss_OID         OID;
    OM_uint32*      integer;
    char*           string;
    uuid_t*         uuid;
    gss_buffer_t    buffer;
    }  gss_value;
```

9.4.1.2 Identifier set

Identifier sets have the following data structure:

```
typedef struct gss_id_set_desc  {
    OM_uint32       id_count;
    gss_id*         ids;
    }  gss_id_set;
```

The id_count field contains the number of Identifiers in the set.

9.4.1.3 Time periods

A time period has the following structure:

```
typedef struct gss_time_period_desc   {
    time_t      start_time;     /* NULL for unconstrained start time */
    time_t      end_time;       /* NULL for unconstrained end time */
    }  gss_time_period;
```

9.4.1.4 time period list

Time period lists have the following data structure:

```
typedef struct gss_period_list _desc{
    OM_uint32               period_count;
    gss_time_period*        periods;
    }  gss_period_list;
```

The period_count field contains the number of time periods in the list.

9.4.1.5 Security attributes

Security attributes (see clause 9.1) have the following data structure:

```
typedef struct gss_sec_attr_desc    {
    gss_OID          attribute_type;
    gss_buffer_t     defining_authority;
                          /* specify GSS_C_NO_BUFFER when non present */
    gss_buffer_t     security_value;
    } gss_sec_attr;
```

Correspondence between the security_value field and the actual syntax of the security attribute is defined along with each specific attribute_type.

9.4.1.6 Security Attribute Sets

A set of security attributes has the following structure:

```
typedef struct gss_sec_attr_set_desc{
    OM_uint32        attribute_count;
    gss_sec_attr*    attributes;
    } gss_sec_attr_set;
```

The attribute_count field contains the number of security attributes in the set.

9.4.1.7 Credentials List

A list of credentials has the following structure:

```
typedef struct    {
    OM_uint32        cred_count;
    gss_cred_id_t*   cred_list;
    } gss_cred_list;
```

The cred_count field contains the number of credentials in the list.

9.4.1.8 Acceptor Control

Acceptor control has the following structure:

```
typedef struct gss_acceptor_control_desc {
    gss_sec_attr     target_only;
                     /* specify GSS_C_NULL_SEC_ATTR when non present */
    gss_sec_attr     delegate_only;
                     /* specify GSS_C_NULL_SEC_ATTR when non present */
    gss_sec_attr     delagate_target;
                     /* specify GSS_C_NULL_SEC_ATTR when non present */
    OM_uint32        delegation_mode;
                     /* specify NULL when non present */
    } gss_acceptor_control;
```

9.4.1.9 Acceptor Control Set

A set of Acceptor Control has the following structure :

```
typedef struct gss_control_set_desc{
    OM_uint32                 control_count;
    gss_acceptor_control*     acceptor_controls;
    } gss_control_set;
```

The control_count field contains the number of acceptor controls in the set.

9.4.2 gss_set_cred_attributes

```
/* set attributes values in credentials */

OM_uint32 gss_set_cred _attributes (
        gss_cred_id_t       cred_handle,              /* IN */
        gss_sec_attr_set    required_attributes,      /* IN */
        OM_uint32           new_cred_req,             /* IN*/
        OM_uint32           commit_cred_req,          /* IN*/
        OM_uint32*          minor_status,             /* OUT*/
        gss_cred_id_t*      output_cred_handle);      /* OUT */
```

9.4.3 gss_get_sec_attributes

```
/* get attributes associated with credentials or security context */

OM_uint32 gss_get_sec_attributes (
        gss_cred_id_t       cred_handle,              /* IN */
        gss_ctx_id_t        context_handle,           /* IN */
        gss_OID_set         attribute_types_required, /* IN */
        OM_uint32*          minor_status,             /* OUT*/
        gss_sec_attr_set**  priv_attributes,          /* OUT*/
        gss_sec_attr_set**  misc_attributes);         /* OUT*/
```

9.4.4 gss_get_received_creds

```
/* get received credentials associated with a security context */

OM_uint32 gss_get_received_creds (
        gss_ctx_id_t        context_handle,           /* IN */
        OM_uint32*          minor_status,             /* OUT*/
        gss_cred_list**     received_creds);          /* OUT*/
```

9.4.5 gss_set_cred_controls

```
/* Set acceptor controls in credentials for context establishmentt */
```

```
OM_uint32 gss_set_cred_controls (
        gss_cred_id_t           cred_handle,            /* IN */
        gss_ control_set        required_control,       /* IN */
        OM_uint32               replace_old_controls,   /* IN */
        OM_uint32               new_cred_req,           /* IN */
        OM_uint32               commit_cred_req,        /* IN */
        OM_uint32*              minor_status,           /* OUT*/
        gss_cred_id_t*          output_cred_handle);    /* OUT*/
```

9.4.6 gss_get_sec_controls

```
/* set context acceptor controls on credentials */

OM_uint32 gss_get_sec_controls (
        gss_cred_id_t           cred_handle,            /* IN */
        gss_ctx_id_t            context_handle,         /* IN */
        OM_uint32*              minor_status,           /* OUT*/
        gss_control_set*        acceptor_controls);     /* OUT*/
```

9.4.7 gss_compound_cred

```
/* compound credentials for delegation */

OM_uint32 gss_compound_cred (
        gss_cred_id_t           delegated_cred_handle,  /* IN */
        gss_cred_id_t           cred_handle,            /* IN */
        OM_uint32*              minor_status,           /* OUT*/
        gss_cred_id_t           cred_handle_new);       /* OUT*/
```

10 Relationship to other standards

This standard further develops the work done in [ECMA-219], defining a security mechanism that supports the GSS-API [GSS-API]. The relationship to ECMA-219 is:

- development of this Standard has resulted in minor corrections to ECMA-219,
- the syntax of the PAC and related data (in support of PAC protection mechanisms), is as defined in ECMA-219,
- the key package syntax is built upon the basic syntax given in ECMA-219, in the manner specified there,
- the attribute structure returned across the interface by the support functions defined in clause 9 is a simplified profile of the structure of the ECMA-219 Security Attribute. This makes the functions more suitable for use by callers of other GSS-API mechanisms.

Other standards related work on similar mechanisms ([KRB5GSS], [SPKM]) has resulted in definitions of GSS-API tokens for those mechanisms. The token formats specified here, have been designed for best fit with the equivalent definitions from these other specifications. In particular:

- the symmetric key distribution schemes make use of Kerberos Tickets (see [Kerberos] and annex B), to which ECMA PAC control information (specifically the PPID) has been added in the authorisation data,
- the asymmetric key distribution schemes make use of Simple Public Key Mechanism tickets, see [SPKM],

- Security Association identifiers and other token protection data have been specified in the same manner as in [SPKM] (called context identifiers there),

The structure of public key certificates, algorithm identifiers and distinguished names used in this Standard conform to [ISO/IEC 9594-8].

Annex A

(Normative)

Formal ASN.1 definitions of data types defined in this standard

This annex contains formal definitions of the ASN.1 data types that are newly defined in this standard.

```
Ecma-gss-api-types    {   iso(1) identified-organisation(3) icdecma(0012) standard(0)
                          ecma-gss-api(235) ecma-gss-api-types (1)

DEFINITIONS ::=

BEGIN

-- exports everything

IMPORTS

    kd-schemes
      FROM    UsefulDefinitions { iso(1) identified-organisation(3) icd-ecma(0012) standard(0)
              apa(219) modules(1) usefulDefinitions(1) }

    Identifier, Signature, Seal, SecurityAttribute, UniqueNumber, TimePeriods, CertAndECV,
    TargetKeyBlock, DialogueKeyBlock
      FROM    SecurityInformationObjects {iso (1) identified-organisation(3)
              icd-ecma(0012) standard(0) apa(219) modules(1)
              securityInformationObjects(3) }

    Certificate, CertificateList, AlgorithmIdentifier, Certificates
      FROM    AuthenticationFramework {joint-iso-ccitt(2) ds(5) module(1)
              authenticationFramework(7) }

    HostAddress, HostAddressList, Ticket
      FROM    Ecma-kerberos-types { iso(1) identified-organisation(3)
              icd-ecma(0012) standard(0) ecma-gss-api(235) ecma-kerberos-types (2)

    SPKM_REQ
      FROM    Ecma-spkm-types { iso(1) identified-organisation(3) icdecma(0012)
              standard(0) ecma-gss-api(235) ecma-spkm-types (3)
```

NOTE
strictly speaking SPKM_REQ does not need to be imported since it is not used explicitly in the syntax given below. However it is one of the possible ANY choices in the TargetKeyBlock construct.

```
-- OBJECT IDENTIFIERS

generic-ecma-mech OBJECT IDENTIFIER ::=
    { iso(1) identified-organisation(3) icd-ecma(0012) standard(0) ecma_gss_api (235)
    generic-ecma-mech (4)}

misc-attribute          OBJECT IDENTIFIER ::= { iso(1) identified-organisation(3) icd-ecma(012) technical-
report(1) security-in-open-systems(046) misc-attribute(3) }

privilege-attribute     OBJECT IDENTIFIER ::= { iso(1) identified-organisation(3) icd-ecma(012) technical-
report(1) security-in-open-systems(046) privilege-attribute(4) }

qualifier-attribute     OBJECT IDENTIFIER ::= { iso(1) identified-organisation(3) icd-ecma(012) technical-
report(1) security-in-open-systems(046) qualifier-attribute(5) }

        -- Key Scheme Object Identifiers

symmIntradomain             OBJECT IDENTIFIER ::= {kd-schemes 1}
symmInterdomain             OBJECT IDENTIFIER ::= {kd-schemes 2}
hybridInterdomain           OBJECT IDENTIFIER ::= {kd-schemes 3}
asymmInitToSymmTarget       OBJECT IDENTIFIER ::= {kd-schemes 4}
symmInitToAsymmTarget       OBJECT IDENTIFIER ::= {kd-schemes 5}
asymmetric                  OBJECT IDENTIFIER ::= {kd-schemes 6}

        -- Object Identifier for insertion in SPKM construct

gss-key-estb-alg AlgorithmIdentifier ::= {kd-schemes, NULL }

-- privilege attribute Object Identifiers

role-type                   OBJECT IDENTIFIER ::= {privilege-attribute   1 }
access-identity-type        OBJECT IDENTIFIER ::= {privilege-attribute   2 }
primary-group-type          OBJECT IDENTIFIER ::= {privilege-attribute   3 }
group-type                  OBJECT IDENTIFIER ::= {privilege-attribute   4 }
role-name-type              OBJECT IDENTIFIER ::= {privilege-attribute   17 }

-- miscellaneous attribute Object Identifiers

acceptor-name-type          OBJECT IDENTIFIER ::= {miscellaneous-attribute      1 }
audit-identity-type         OBJECT IDENTIFIER ::= {miscellaneous-attribute      2 }
issuer-domain-name-type     OBJECT IDENTIFIER ::= {miscellaneous-attribute      10 }
validity-periods-type       OBJECT IDENTIFIER ::= {miscellaneous-attribute      11 }
optional-restrictions-type  OBJECT IDENTIFIER ::= {miscellaneous-attribute      12 }
mandatory-restrictions-type OBJECT IDENTIFIER ::= {miscellaneous-attribute      13 }
```

```
-- qualifier attribute Object Identifiers

acceptor-name-type          OBJECT IDENTIFIER ::= {qualifier-attribute          1 }
application-trust-group-type OBJECT IDENTIFIER ::= {qualifier-attribute          2 }

-- data types

AcceptorControl ::= SEQUENCE {
    targetOnly              SEQUENCE OF SecAttribute OPTIONAL,
    delegateOnly            SEQUENCE OF SecAttribute OPTIONAL,
    delegateTarget          SEQUENCE OF SecAttribute OPTIONAL,
    delegationMode          DelegationMode    OPTIONAL,
                                }

CDTContents ::= SEQUENCE {
    tokenType               [0]    OCTET STRING VALUE X'0301',
    SAId                    [1]    OCTET STRING,
    utcTime                 [2]    UTCTime OPTIONAL,
    usec                    [3]    INTEGER OPTIONAL,
    seq-number              [4]    INTEGER OPTIONAL
                             }

ContextDeleteToken ::=  SEQUENCE  {
    cdtContents    [0] CDTContents,
    cdtSeal        [1] Seal         -- seal over cdtContents, encrypted
                                    -- under the Integrity Dialogue Key
                                    -- contains only the sealValue field
                               }

DelegationMode  :=    ENUMERATED {
                            default             (0),
                            simple              (1),
                            traced              (2),
                            composite           (3)     }

ECMA-AUTHORISATION-DATA-TYPE ::= INTEGER { ECMA-ADATA (65) }
ECMA-AUTHORISATION-DATA ::= SEQUENCE {
    ecma-ad-type      [0] ENUMERATED    { ppidType   (0)},
    ecma-ad-value     [1] CHOICE        { ppidValue  [0]       SecurityAttribute}}
                                  -- only one choice for now

EcmaMechSpecInfo ::= SEQUENCE OF Attribute
```

```
ErrorArgument ::= ENUMERATED  {
      gss_ecma_s_sg_server_sec_assoc_open                    (1),
      gss_ecma_s_sg_ incomp_cert_syntax                      (2),
      gss_ecma_s_sg_ bad_cert_attributes                     (3),
      gss_ecma_s_sg_ inval_time_for_attrib                   (4),
      gss_ecma_s_sg_ pac_restrictions_prob                   (5),
      gss_ecma_s_sg_ issuer_problem                          (6),
      gss_ecma_s_sg_ cert_time_too_early                     (7),
      gss_ecma_s_sg_ cert_time_expired                       (8),
      gss_ecma_s_sg_ invalid_cert_prot                       (9),
      gss_ecma_s_sg_ revoked_cert                            (10),
      gss_ecma_s_sg_ key_constr_not_supp                     (11),
      gss_ecma_s_sg_ init_kd_server_ unknown                 (12),
      gss_ecma_s_sg_init_unknown                             (13),
      gss_ecma_s_sg_alg_problem_in_dialogue_key_block        (14),
      gss_ecma_s_sg_no_basic_key_for_dialogue_key_block      (15),
      gss_ecma_s_sg_key_distrib_prob                         (16),
      gss_ecma_s_sg_invalid_user_cert_in_key_block           (17),
      gss_ecma_s_sg_unspecified                              (18),
      gss_ecma_s_sg_invalid_token_format                     (19)
                                        }

ErrorToken ::=   {
         tokenType       [0]     OCTET STRING VALUE X'0400',
         etContents      [1]     ErrorArgument,
                  }

HashedNameInput ::= SEQUENCE {
      hniPlainKey         [0] BIT STRING,    -- the same value as plainKey
      hniIssuingKDS       [1] Identifier}
```

```
ICTContents ::= SEQUENCE {
    tokenId                 [0]   INTEGER,       -- shall contain X'0100'
    SAId                    [1]   OCTET STRING,
    targetAEFPart           [2]   TargetAEFPart,
    targetAEFPartSeal       [3]   Seal,  -- Imported from [ECMA-219] (see annex E)
    contextFlags            [4]   BIT STRING{
                                      delegation          (0),
                                      mutual-auth         (1),
                                      replay-detect       (2),
                                      sequence            (3),
                                      conf-avail          (4),
                                      integ-avail         (5)
                                        }
    utcTime                 [5]   UTCTime OPTIONAL,
    usec                    [6]   INTEGER OPTIONAL,
    seq-number              [7]   INTEGER OPTIONAL,
    initiatorAddress        [8]   HostAddress OPTIONAL,     -- imported from [Kerberos]
    targetAddress           [9]   HostAddress OPTIONAL      -- imported from [Kerberos]
                                                            -- used as channel bindings    }

InitialContextToken ::=  SEQUENCE    {
    ictContents     [0] ICTContents,
    ictSeal         [1] Seal   -- Imported from [ECMA-219] (see annex E)
                                                        }

KeyEstablishmentData ::= SEQUENCE {
        encryptedPlainKey   [0] BIT STRING,         -- encrypted PlainKey
        targetName          [1] Identifier          OPTIONAL,
        nameHashingAlg      [2] AlgorithmIdentifier OPTIONAL}

MICToken  ::=   PMToken

PlainKey ::= SEQUENCE {
    plainKey        [0] BIT STRING,         -- The cleartext key
    hashedName      [1] BIT STRING}
```

```
PMTContents ::= SEQUENCE {
    tokenId                 [0]   INTEGER,
    SAId                    [1]   OCTET STRING,
    seq-number              [2]   INTEGER                          OPTIONAL,
    userData                [3]   CHOICE {
                                      plaintext       BIT STRING,
                                      ciphertext      OCTET STRING
                                          }                        OPTIONAL,
    directionIndicator      [4]   BOOLEAN                          OPTIONAL
                                  }

PMToken ::=  SEQUENCE    {
    pmtContents       [0] PMTContents,
    pmtSeal           [1] Seal
                          }

PublicKeyBlock ::= SEQUENCE{
    signedPKBPart[0]     SignedPKBPart,
    signature         [1]     Signature                            OPTIONAL,
                                  -- imported from [ECMA-219], see also annex E
    certificate       [2]     Certificate                          OPTIONAL}
                                  -- imported from [ISO/IEC 9594-8], see also annex E

PublicTicket ::= SEQUENCE{
    krb5Ticket        [0]     Ticket,                  -- see annex B
    publicKeyBlock    [1]     PublicKeyBlock}

SecAttribute ::= {
    attributeType       OBJECT IDENTIFIER,
    definingAuthority   OCTET STRING                               OPTIONAL,
    securityValue       OCTET STRING }

SignedPKBPart ::= SEQUENCE{
    keyEstablishmentData      [0]     KeyEstablishmentData,
    encryptionMethod          [1]     AlgorithmIdentifier          OPTIONAL,
    issuingKDS                [2]     Identifier,
    uniqueNumber              [3]     UniqueNumber,
    validityTime              [4]     TimePeriods,
                                          -- UniqueNumber and TimePeriods are
                                          -- imported from [ECMA-219]
    creationTime              [5]     UTCTime}
```

```
TargetAEFPart ::= SEQUENCE  {
    pacAndCVs           [0] SEQUENCE OF CertandECV OPTIONAL,
    targetKeyBlock      [1] TargetKeyBlock,    -- see clause 5 for schemes supported
    dialogueKeyBlock    [2] DialogueKeyBlock, -- Imported from [ECMA-219] (see annex E)
    targetIdentity      [3] Identifier,        -- Imported from [ECMA-219] (see annex E)
    flags               [4] BIT STRING{
                                    delegation          (0)
                                    }
                        }

TargetKeyBlock ::= SEQUENCE {
    kdSchemeOID     [2] OBJECT IDENTIFIER,
    targetKDSpart   [3] ANY                 OPTIONAL,
                                            -- depending on kdSchemeOID
    targetPart      [4] ANY                 OPTIONAL}
                                            -- depending on kdSchemeOID

TargetResultToken ::=  SEQUENCE    {
    trtContents     [0] TRTContents,
    trtSeal         [1] Seal            }

Token ::=
    [APPLICATION 0] IMPLICIT SEQUENCE {
      thisMech              MechType,   -- the OBJECT IDENTIFIER specified below
      innerContextToken ANY DEFINED BY thisMech }

TRTContents ::= SEQUENCE {
    tokenId             [0]     INTEGER,        -- shall contain X'0200'
    SAId                [1]     OCTET STRING,
    utcTime             [5]     UTCTime OPTIONAL,
    usec                [6]     INTEGER OPTIONAL,
    seq-number          [7]     INTEGER OPTIONAL
                        }

WrapToken  ::=   PMToken

END -- of Ecma-gss-api-types
```

Annex B

(Normative)

Definitions of [Kerberos] data types

This annex contains formal normative ASN.1 specifications for data types that are also defined in [Kerberos]. Where any differences might arise, the specifications in this annex take precedence.

```
Ecma-kerberos-types   {   iso(1) identified-organisation(3) icd-ecma(0012) standard(0)
                          ecma-gss-api(235) ecma-kerberos-types (2)

DEFINITIONS ::=

BEGIN

-- exports everything

IMPORTS

-- imports nothing

-- data types

AuthorizationData ::= SEQUENCE OF SEQUENCE {
     ad-type       [0]     INTEGER,
     ad-data       [1]     OCTET STRING}

EncryptedData ::= SEQUENCE {
     etype         [0] INTEGER, -- EncryptionType
     kvno          [1]     INTEGER                 OPTIONAL,
     cipher        [2]     OCTET STRING    —ciphertext}

EncryptionKey ::= SEQUENCE {
     keytype       [0]     INTEGER,
     keyvalue      [1]     OCTET STRING}
```

```
EncTicketPart ::= [APPLICATION 3] SEQUENCE {
      flags                [0]    TicketFlags,
      key                  [1]     EncryptionKey,
      crealm               [2]    Realm,
      cname                [3]    PrincipalName,
      transited            [4]    TransitedEncoding,
      authtime             [5]    KerberosTime,
      starttime            [6]    KerberosTime OPTIONAL,
      endtime              [7]    KerberosTime,
      renew-till           [8]    KerberosTime OPTIONAL,
      caddr                [9]    HostAddresses OPTIONAL,
      authorization-data   [10]   AuthorizationData OPTIONAL}
         -- see clause 5.3.5 for the specific ECMA contents of AuthorizationData

HostAddress ::= SEQUENCE {
      addr-type            [0]    INTEGER,
      address              [1]    OCTET STRING}

HostAddresses ::= SEQUENCE OF SEQUENCE {
      addr-type            [0]    INTEGER,
      address              [1]    OCTET STRING}

KerberosTime ::= GeneralizedTime
         -- Specifying UTC time zone (Z)

PrincipalName ::= SEQUENCE {
      name-type            [0]    INTEGER,
      name-string          [1]    SEQUENCE OF GeneralString}

Realm ::= GeneralString

Ticket ::= [APPLICATION 1] SEQUENCE {
      tkt-vno              [0]    INTEGER,
      realm                [1]    Realm,
      sname                [2]    PrincipalName,
      enc-part             [3]    EncryptedData} -- decrypts to EncTicketPart
```

```
TicketFlags ::= BIT STRING {
    reserved            (0),    -- not supported in the ECMA mechanism
    forwardable         (1),    -- not supported in the ECMA mechanism
    forwarded           (2),    -- not supported in the ECMA mechanism
    proxiable           (3),    -- not supported in the ECMA mechanism
    proxy               (4),    -- not supported in the ECMA mechanism
    may-postdate        (5),    -- not supported in the ECMA mechanism
    postdated           (6),
    invalid             (7),    -- not supported in the ECMA mechanism
    renewable           (8),    -- not supported in the ECMA mechanism
    initial             (9),    -- not supported in the ECMA mechanism
    pre-authent         (10),
    hw-authent          (11)}

TransitedEncoding ::= SEQUENCE {
    tr-type             [0]     INTEGER, -- must be registered
    contents            [1]     OCTET STRING}
        -- the TransitedEncoding construct is not used in the ECMA mechanism.

END -- of Ecma-kerberos-types
```

Annex C

(Normative)

Definitions of [SPKM] data types

This annex contains formal normative ASN.1 specifications for data types that are also defined in [SPKM]. Where any differences might arise, the specifications in this annex take precedence.

```
Ecma-spkm-types   {   iso(1) identified-organisation(3) icdecma(0012) standard(0)
                      ecma-gss-api(235) ecma-spkm-types (3)

DEFINITIONS ::=

BEGIN

-- exports everything

IMPORTS

    AuthorizationData
      FROM Ecma-kerberos-types   {
                      iso(1) identified-organisation(3) icd-ecma(0012) standard(0)
                      ecma-gss-api(235) ecma-kerberos-types (2)

    AlgorithmIdentifier, Certificate, CertificateList, CertificatePair
      FROM AuthenticationFramework {
                      joint-iso-ccitt ds(5) modules(1) authenticationFramework(7) }

    Name
      FROM InformationFramework {
                      joint-iso-ccitt ds(5) modules(1) informationFramework(1) }

-- data types

CertificationData ::= SEQUENCE {
    certificationPath          [0] CertificationPath      OPTIONAL,
    certificateRevocationList  [1] CertificateList        OPTIONAL
}   -- at least one of the above shall be present
```

```
CertificationPath ::= SEQUENCE    {
    userKeyId           [0]    OCTET STRING                    OPTIONAL,
                                    -- identifier for user's public key
    userCertif          [1]    Certificate                     OPTIONAL,
                                    -- certificate containing user's public key
    verifKeyId          [2]    OCTET STRING                    OPTIONAL,
                                    -- identifier for user's public verification key
    userVerifCertif     [3]    Certificate                     OPTIONAL,
                                    -- certificate containing user's public verification key
    theCACertificates   [4]    SEQUENCE OF CertificatePair     OPTIONAL
                                    -- certification path from target to source
                                }

ChannelId ::= OCTET STRING

Conf_Algs ::= CHOICE{
    SEQUENCE OF AlgorithmIdentifier,
    NULL                                -- used when conf. is not available over context
                        }               -- for C-ALG

Context_Data ::= SEQUENCE    {
    channelId           ChannelId,              -- channel bindings
    seq_number          INTEGER OPTIONAL,       -- sequence number
    options             Options,
    conf_alg            Conf_Algs,              -- confidentiality. algs.
    intg_alg            Intg_Algs               -- integrity algorithm
                            }

ENCRYPTED MACRO ::=
BEGIN
TYPE NOTATION       ::= type(ToBeEnciphered)
VALUE NOTATION::= value(VALUE BIT STRING)
END     -- of ENCRYPTED

HASHED MACRO ::=
BEGIN
    TYPE  NOTATION    ::= type       ( ToBeHashed )
    VALUE NOTATION    ::= value      ( VALUE OCTET STRING )
END -- hash used is the one specified for the MANDATORY I-ALG

Intg_Algs ::= SEQUENCE OF AlgorithmIdentifier  -- for I-ALG

Key_Estb_Algs ::= SEQUENCE OF AlgorithmIdentifier  -- to allow negotiation of K-ALG
```

```
MAC MACRO ::=
BEGIN
      TYPE  NOTATION ::= type  ( ToBeMACed )
      VALUE NOTATION ::= value    ( VALUE
                        SEQUENCE{
                              algId       AlgorithmIdentifier,
                              mac         BIT STRING
                                          }
                                          )
END

Options ::= BIT STRING   {
      delegation_state                (0),
      mutual_state                    (1),
      replay_det_state                (2),        -- used for replay det. during context
      sequence_state                  (3),        -- used for sequencing during context
      conf_avail                      (4),
      integ_avail                     (5),
      target_certif_data_required     (6)         -- used to request targ's certif. data
                              }

Random_Integer ::= BIT STRING

Req_Integrity ::= CHOICE {
      sig_integ         [0] SIGNATURE REQ_TOKEN,
      mac_integ         [1] MAC REQ_TOKEN
                                  }
```

```
REQ_TOKEN ::= SEQUENCE {
    tok_id              INTEGER,                -- shall contain 0100 (hex)
    context_id          Random_Integer,
    pvno                BIT STRING,             -- protocol version number
    timestamp           UTCTime                         OPTIONAL,
                                                -- mandatory for SPKM-2
    randSrc             Random_Integer,
    targ_name           Name,
    src_name            Name,                   -- may be a value indicating "anonymous"
    req_data            Context_Data,
    validity        [0] Validity                        OPTIONAL,
                                                -- validity interval for key (may be used in the
                                                -- computation of security context lifetime)
    key_estb_set    [1] Key_Estb_Algs,          -- specifies set of key establishment algorithms
    key_estb_req        BIT STRING                      OPTIONAL,
                            -- key estb. parameter corresponding to first K-ALG in set
                            -- (not used if initiator is unable or unwilling to
                            -- generate and securely transmit key material to target).
                            -- Established key must be sufficiently long to be used
                            -- with any of the offered confidentiality algorithms.
    key_src_bind        HASHED SEQUENCE {
                            src_name        Name,
                            symm_key        BIT STRING}    OPTIONAL
                            -- used to bind the source name to the symmetric key
                            -- (i.e., the unprotected version of what is
                            -- transmitted in key_estb_req).
    }

SIGNATURE MACRO ::=
BEGIN
TYPE NOTATION   ::= type (OfSignature)
VALUE NOTATION  ::= value(VALUE
    SEQUENCE    {
        AlgorithmIdentifier,
        ENCRYPTED OCTET STRING
                }
                        )
END
```

```
SPKM_REQ ::= SEQUENCE {
      requestToken              REQ_TOKEN,
      req_integrity             Req_Integrity,
      certif_data         [2]   CertificationData OPTIONAL,
      auth_data           [3]   AuthorizationData OPTIONAL
         -- see [Kerberos] for a discussion of authorization data
}

Validity ::= SEQUENCE {
                  notBefore       UTCTime,
                  notAfter        UTCTime }

END -- of Ecma-spkm-types
```

Annex D

(Normative)

Mappings of Minor Status Returns onto [ECMA-219] error values

This annex contains a table showing how the minor status returns defined in clause 8 correspond to the error values of [ECMA-219]. Many of the minor_status codes at the API level cover more than one of the errors from the ECMA Mechanism, since any mechanism specific operation, argument or result has a higher granularity and variety of parameters than the GSS-API.

GSS_API parameter	**ECMA-219 Error**			
Minor Status code	ABSTRACT-ERROR	Parameter	Name	Val
GSS_ECMA_S_SG_SERVER_SA_ ALREADY_ESTABLISHED	AssocAlreadyOpen	NULL	n/a	n/a
GSS_ECMA_S_SG_ INCOMP_CERT_SYNTAX	AUCSpecificError	AUCProblem	incompatible SyntaxVersion	1
	CertificateError	CertificateProblem	incompatible SyntaxVersion	1
	PACSpecificError	PACProblem	incompatible SyntaxVersion	2
GSS_ECMA_S_SG_ BAD_CERT_ATTRIBUTES	AUCSpecificError	AUCProblem	invalidProtection	2
	AUCSpecificError	AUCProblem	wrongType	3
	AUCSpecificError	AUCProblem	authentication Level	4
	AUCSpecificError	AUCProblem	invalidAttribute	5
	PACSpecificError	PACProblem	historyNot Acceptable	1
	PACSpecificError	PACProblem	invalidMisc Attributes	3
	PACSpecificError	PACProblem	invalidPrivilegeAttributes	4
	PACSpecificError	PACProblem	invalidProtection	5
	PACSpecificError	PACProblem	invalidTraceLink	7
	PACSpecificError	PACProblem	wrongType	9
GSS_ECMA_S_SG_ INVAL_TIME_FOR_ATTRIB	AUCSpecificError	AUCProblem	invalidTime	6
	PACSpecificError	PACProblem	invalidTime	6
GSS_ECMA_S_SG_ PAC_RESTRICTIONS_PROB	PACSpecificError	PACProblem	restrictionType NotSupported	8
GSS_ECMA_S_SG_ISSUER_PROBLEM	CertificateError	CertificateProblem	issueProblem	2
GSS_ECMA_S_SG_ CERT_TIME_TOO_EARLY	CertificateError	CertificateProblem	timeProblem	3
GSS_ECMA_S_SG_ CERT_TIME_EXPIRED	CertificateError	CertificateProblem	certificateExpire	12
GSS_ECMA_S_SG_ INVALID_CERT_PROT	CertificateError	CertificateProblem	invalidCheck ValueType	4
	CertificateError	CertificateProblem	invalidCheck ValueTarget	5
	CertificateError	CertificateProblem	invalidSeal	6
	CertificateError	CertificateProblem	invalidSignature	7
	CertificateError	CertificateProblem	invalidSymme-tricAlgId	8
	CertificateError	CertificateProblem	invalidAsymme-tricAlgId	9
	CertificateError	CertificateProblem	invalidHash AlgId	10
GSS_ECMA_S_SG_ REVOKED_CERT	CertificateError	CertificateProblem	certificate Revoked	11
GSS_ECMA_S_SG_ KEY_CONSTR_NOT_SUPP	ConstructTypes NotSupported	NULL	n/a	n/a
GSS_ECMA_S_SG_ INIT_KD_SERVER_ UNKNOWN	InitiatorKDS Unknown	NULL	n/a	n/a
GSS_ECMA_S_SG_ INIT_UNKNOWN	InitiatorUnknown	NULL	n/a	n/a
GSS_ECMA_S_SG_ INSUFF_AUTHORISATION	Insufficient Authorisation	NULL	n/a	n/a

GSS_API parameter	ECMA-219 Error			
Minor Status code	ABSTRACT-ERROR	Parameter	Name	Val
GSS_ECMA_S_SG_ALG_PROBLEM_ IN_ DIALOGUE_KEY_BLOCK	InvalidDialogueKeyBlock	useAlgorithm NotSupported	n/a (ENUMERATED)	1
	InvalidDialogueKeyBlock	useHashAlg NotSupported	n/a (ENUMERATED)	2
	InvalidDialogueKeyBlock	derivationAlgorithmNotSupported	n/a (ENUMERATED)	3
GSS_ECMA_S_SG_ NO_BASIC_KEY_FOR_ DIALOGUE_KEY_BLOCK	InvalidDialogueKeyBlock	noBasicKey	n/a (ENUMERATED)	4
GSS_ECMA_S_SG_ KEY_DISTRIB_PROB	InvalidKeyBlock	kdScheme NotSupported	n/a (ENUMERATED)	1
	InvalidKeyBlock	invalidTargetPart	n/a (ENUMERATED)	2
	KDScheme NotSupported	NULL	n/a	n/a
	KiType NotSupported	NULL	n/a	n/a
GSS_ECMA_S_SG_INVALID_USER_ CERT_ IN_KEY_BLOCK	InvalidKeyBlock	userCertificate Error	n/a (ENUMERATED)	3
GSS_ECMA_S_SG_ OPERATION_NOT_SUPP	Operation NotSupported	NULL	n/a	n/a
GSS_ECMA_S_SG_ SEC_ASSOC_ID_FAILURE	Security Association Failure	NULL	n/a	n/a
GSS_ECMA_S_SG_ UNACCEPTABLE_ACT_REQ	UnacceptableAttribute-Requirements	unacceptable Refinement	n/a (ENUMERATED)	1
	UnacceptableAttribute-Requirements	unacceptable AttributeSet	n/a (ENUMERATED)	2
	UnacceptableAttribute-Requirements	unacceptable AttributeValues	n/a (ENUMERATED)	3
	Unacceptable-Restriction-Requirements	algId	OBJECT IDENTIFIER	n/a
	Unacceptable-TicketRequirements	unacceptable CertificateType	n/a (ENUMERATED)	1
	Unacceptable-TicketRequirements	unacceptable ProtectionType	n/a (ENUMERATED)	2
	Unacceptable-TicketRequirements	unacceptable TargetInformation	n/a (ENUMERATED)	3
	Unacceptable-TicketRequirements	unacceptable CertificateControls	n/a (ENUMERATED)	4
GSS_ECMA_S_SG_ UNSPECIFIED	Unspecified	NULL	n/a	n/a

Table 8 - Relationship between minor status returns and ECMA-219 errors

Annex E

(Informative)

Imported Types

This annex contains, for completeness, the relevant ASN.1 types imported from other standards.

E.1 Types imported from [ECMA-219]

The syntax below has been taken directly from [ECMA-219]. It contains the ECMA syntax elements needed to understand this Standard, and the value of the kd-schemes OBJECT IDENTIFIER. It is provided for information only. The definitive syntax is that in [ECMA-219]. Not all of the internal fields of all of the constructs are expanded here. The constructs are listed in alphabetical order.

```
CertandECV ::=      SEQUENCE {
    certificate                 [0]     GeneralisedCertificate,
    ecv                         [1]     ECV,                    OPTIONAL}

CertificateBody ::= CHOICE{
    encryptedBody               [0]     BIT STRING,
    normalBody                  [1]     SEQUENCE{
                                            commonContents  [0]     CommonContents,
                                            specificContents    [1]     SpecificContents}}

CertificateId ::=SEQUENCE {     issuerDomain    [0]     Identifier      OPTIONAL,
                                issuerIdentity  [1]     Identifier,
                                serialNumber    [2]     INTEGER}
                                    -- serialNumber is the same as in [ISO/IEC 9594-8]

CheckValue ::= CHOICE  {        signature       [0]     Signature,
                                seal            [1]     Seal,
                                unprotected     [2]     NULL}

CommonContents ::= SEQUENCE{
    comConSyntaxVersion     [0]     INTEGER { version1 (1) }            DEFAULT 1,
    issuerDomain            [1]     Identifier                          OPTIONAL,
    issuerIdentity          [2]     Identifier,
    serialNumber            [3]     INTEGER,        --as structured in [ISO/IEC 9594-8]
    creationTime            [4]     UTCTime                             OPTIONAL,
    validity                [5]     Validity,
    algId                   [6]     AlgorithmIdentifier,
    hashAlgId               [7]     AlgorithmIdentifier                 OPTIONAL}
--  AlgorithmIdentifier is imported from [ISO/IEC 9594-8]

CValues ::= SEQUENCE OF SEQUENCE {
    index               [0]     INTEGER,
    value               [1]     BIT STRING}
```

```
DialogueKeyBlock      ::=     SEQUENCE {
        integKeySeed              [0]     SeedValue,
        confKeySeed               [1]     SeedValue,
        integKeyDerivationInfo    [2]     KeyDerivationInfo    OPTIONAL,
        confKeyDerivationInfo     [3]     KeyDerivationInfo    OPTIONAL,
        integDKuseInfo            [4]     DKuseInfo            OPTIONAL,
        confDKuseInfo             [5]     DKuseInfo            OPTIONAL }

        DKuseInfo           ::= SEQUENCE {
            useAlgId              [0]     AlgorithmIdentifier,
            useHashAlgId          [1]     AlgorithmIdentifier  OPTIONAL }

ECV ::= SEQUENCE   {
        crypAlgIdentifier         [0]     AlgorithmIdentifier  OPTIONAL,
                                      --  AlgorithmIdentifier is imported from [ISO/IEC 9594-8]
        cValues                   [1]     CHOICE {
                                              encryptedCvalueList  [0]     BIT STRING,
                                              individualCvalues    [1]     CValues }
                        }

GeneralisedCertificate ::= SEQUENCE{
     certificateBody              [0]     CertificateBody,
     checkValue                   [1]     CheckValue}

Identifier ::= CHOICE   {
     objectId                     [0]     OBJECT IDENTIFIER,
     directoryName                [1]     Name,    -- imported from the Directory Standard
     printableName                [2]     PrintableString,
     octets                       [3]     OCTET STRING,
     intVal                       [4]     INTEGER,
     bits                         [5]     BIT STRING,
     pairedName                   [6]     SEQUENCE{
                                              printableName        [0]     PrintableString,
                                              uniqueName           [1]     OCTET STRING}
                        }

KeyConstructionData ::=SEQUENCE{
        constructionMethodId      [0]     OBJECT IDENTIFIER       OPTIONAL,
        methodParameters          [1]     MethodParameters}

KeyDerivationInfo   ::= SEQUENCE {
            owfId                 [0]     AlgorithmIdentifier,
            keySize               [1]     INTEGER }
```

```
PACSpecificContents ::= SEQUENCE{
    pacSyntaxVersion    [0]   INTEGER{   version1 (1)}  DEFAULT 1,
    pacHistory          [1]   SEQUENCE OF CertificateId       OPTIONAL,
    protectionMethods   [2]   SEQUENCE OF MethodGroup         OPTIONAL,
    traceLink           [3]   SEQUENCE OF SecurityAttribute   OPTIONAL,
    pacType             [4]   ENUMERATED{   primaryPrincipal              (1),
                                            temperedSecPrincipal          (2),
                                            untemperedSecPrincipal        (3)}
                                                          DEFAULT 3,
    privileges          [5]   SEQUENCE OF PrivilegeAttribute,
    restrictions        [6]   SEQUENCE OF Restriction         OPTIONAL,
    miscellaneousAtts   [7]   SEQUENCE OF SecurityAttribute   OPTIONAL,
    timePeriods         [8]   TimePeriods                     OPTIONAL}

PrivilegeAttribute ::= SecurityAttribute

PValue ::= SEQUENCE    {
    pv                  [0]   BIT STRING,
    algorithmIdentifier [1]   AlgorithmIdentifier         OPTIONAL }
                -- AlgorithmIdentifier is Imported from [ISO/IEC 9594-8]; Default is MD5

Seal ::= SEQUENCE{
    sealValue           [0]   BIT STRING,
    symmetricAlgId      [1]   AlgorithmIdentifier        OPTIONAL,
    hashAlgId           [2]   AlgorithmIdentifier        OPTIONAL,
    targetName          [3]   Identifier                  OPTIONAL,
    keyId               [4]   INTEGER                OPTIONAL}
--  AlgorithmIdentifier is imported from [ISO/IEC 9594-8]

SecurityAttribute ::= SEQUENCE{
      attributeType     Identifier,
      attributeValue    SET OF SEQUENCE{
                            definingAuthority   [0] Identifier          OPTIONAL,
                            securityValue       [1] SecurityValue} }

      -- NOTE: SecurityAttribute is not tagged, for compatibility with the Directory
               Standard.
```

```
SecurityValue ::= CHOICE{
    directoryName           [0]   Name,    -- imported from the Directory Standard
    printableName           [1]   PrintableString,
    octets                  [2]   OCTET STRING,
    intVal                  [3]   INTEGER,
    bits                    [4]   BIT STRING,
    any                     [5]   ANY}     -- defined by attributeType

SeedValue       ::= SEQUENCE {
      timeStamp             [0]   UTCTime              OPTIONAL,
      random                [1]   BIT STRING }

Signature ::= SEQUENCE{
    signatureValue          [0]   BIT STRING,
    asymmetricAlgId         [1]   AlgorithmIdentifier    OPTIONAL,
    hashAlgId               [2]   AlgorithmIdentifier    OPTIONAL,
    issuerCAName            [3]   Identifier             OPTIONAL,
    caCertInformation       [4]   CHOICE {
                caCertSerialNumber      [0]   INTEGER,
                certificationPath       [1]   CertificationPath}  OPTIONAL}

SpecificContents ::= CHOICE{
    auc                     [0]   AUCSpecificContents,
    pac                     [1]   PACSpecificContents,
    proxyOnlyPAC            [2]   PACSpecificContents,
    initiatorOnlyPAC        [3]   PACSpecificContents,
    gucType                 [4]   GUCSpecificContents,
    otherType               [5]   SEQUENCE {
                                      typeId            [0]  OBJECT IDENTIFIER,
                              otherSpecificContents     [1]  ANY} -- defined by typeId

TargetKeyBlock ::= SEQUENCE {
      initiatorKDSname      [0]   Identifier             OPTIONAL
      kdSchemeOID           [2]   OBJECT IDENTIFIER,
      targetKDSpart         [3]   ANY                    OPTIONAL,
                                                         -- depending on kdSchemeOID
      targetPart            [4]   ANY                    OPTIONAL}
                                                         -- depending on kdSchemeOID

TargetName ::= Identifier

TimePeriods ::= SEQUENCE OF SEQUENCE {
      startTime             [0]   UTCTime                OPTIONAL,
      endTime               [1]   UTCTime                OPTIONAL}
```

```
UniqueNumber  ::= SEQUENCE{
        time                    UTCTime,
        random                  INTEGER         OPTIONAL}

Validity ::=SEQUENCE {
        notBefore               UTCTime,
        notAfter                UTCTime} -- as in [ISO/IEC 9594-8]
```

E.2 Types imported from ISO Frameworks

The data types listed below have been imported from the InformationFramework ([ISO/IEC 9594-2]) and AuthenticationFramework ([ISO/IEC 9594-8]). They are listed in alphabetical order. They are presented here for information only. In the event of a difference from the types identified in ([ISO/IEC 9594-2] or [ISO/IEC 9594-8], the referenced documents take precedence.

```
AlgorithmIdentifier ::= SEQUENCE  {
        algorithm           OBJECT IDENTIFIER,
        parameter           ANY DEFINED BY algorithm OPTIONAL
                                }

AttributeType               ::= OBJECT IDENTIFIER

AttributeValue              ::= ANY

AttributeValueAssertion     ::= SEQUENCE {AttributeType,AttributeValue}

Certificate ::= SIGNED SEQUENCE{
    version                 [0]   Version DEFAULT v1,
    serialNumber            [1]   CertificateSerialNumber,
    signature               [2]   AlgorithmIdentifier,
    issuer                  [3]   Name,
    validity                [4]   Validity,
    subject                 [5]   Name,
    subjectPublicKeyInfo    [6]   SubjectPublicKeyInfo,
    issuerUID               [7]   IMPLICIT UID          OPTIONAL,
    subjectUID              [8]   IMPLICIT UID          OPTIONAL
                                    }
```

```
CertificateList ::= SIGNED SEQUENCE    {
    signature                [0]    AlgorithmIdentifier,
    issuer                   [1]    Name,
    thisUpdate               [2]    UTCTime,
    nextUpdate               [3]    UTCTime              OPTIONAL,
    revokedCertificates      [4]    SEQUENCE OF SEQUENCE {
                                        userCertificate      CertificateSerialNumber,
                                        revocationDate       UTCTime
                                                                    }       OPTIONAL
                                    }

CertificatePair ::= SEQUENCE {
    forward [0]          Certificate OPTIONAL,
    reverse [1]          Certificate OPTIONAL
    } -- at least one of the pair shall be present

CertificateSerialNumber   ::= INTEGER

DistinguishedName             ::= RDNSequence

kd-schemes OBJECT IDENTIFIER ::=
                    {   iso(1) identified-organisation(3) icd-ecma (0012) standard (0)
                        apa(219) 5 }

Name                          ::= CHOICE {      --only one for now
                                      DistinguishedName
                                          }

RDNSequence                   ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName     ::= SET OF AttributeValueAssertion

SIGNED MACRO    ::=
BEGIN
TYPE NOTATION       ::= type (ToBeSigned)
VALUE NOTATION::= value  (VALUE
SEQUENCE{
    ToBeSigned,
    AlgorithmIdentifier,              --of the algorithm used to generate the signature
    ENCRYPTED OCTET STRING    --where the octet string is the
                                      --result of the hashing of the
                                      --value of "ToBeSigned"
    }
                                  )
END                                   -- of SIGNED
```

```
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm           AlgorithmIdentifier,
    subjectPublicKey    BIT STRING
    }

UID ::= BIT STRING
```

Printed copies can be ordered from:

ECMA
114 Rue du Rhône
CH-1204 Geneva
Switzerland

Fax: +41 22 849.60.01
Internet: helpdesk@ecma.ch

Files can be downloaded from our FTP site, **ftp.ecma.ch,** logging in as **anonymous** and giving your E-mail address as **password**. This Standard is available from library **ECMA-ST** as a compacted, self-expanding file in MSWord 6.0 format (file E235-DOC.EXE) as a compacted, self-expanding PostScript file (file E235-PSC.EXE) and as an Acrobat file (file ECMA-235.PDF). File E235-EXP.TXT gives a short presentation of the Standard.

The ECMA site can be reached also via a modem. The phone number is +41 22 735.33.29, modem settings are 8/n/1. Telnet (at ftp.ecma.ch) can also be used.

Our web site, http://www.ecma.ch, gives full information on ECMA, ECMA activities, ECMA Standards and Technical Reports.

ECMA

114 Rue du Rhône
CH-1204 Geneva
Switzerland

This Standard ECMA-235 is available free of charge in printed form and as a file.

See inside cover page for instructions