Internet-Draft
IETF Common Authentication Technology WG
Expire in six months
<draft-ietf-cat-sesamemech-01.txt>

ERIC BAIZE, BULL
STEPHEN FARRELL, SSE
TOM PARKER, ICL

November 22, 1996

The SESAME V5 GSS-API Mechanism

STATUS OF THIS MEMO

ABSTRACT

This specification defines protocols, data elements, and conventions to be employed by peers implementing the Generic Security Service Application Program Interface (as specified in RFCs 1508 and 1509) when using the SESAME Version 5 Mechanism.

1. BACKGROUND

Although the Kerberos Version 5 GSS-API mechanism [KRB5] is becoming well-established in many environments, it is important in some applications to have a GSS-API mechanism, which is able to support privileges rather than only a single identity, which is scalable because it supports public key technology and which is flexible in a distributed environment due to its fine granularity delegation properties.

The mechanism described in this specification has been designed to provide the following features.

1) SESAME allows both unilateral and mutual authentication to be accomplished using loosely synchronous clocks. One key advantage of this feature is that, when unilateral authentication is used, no additional message (as in a challenge-response mechanism) is needed and thus it is possible to concatenate in a single message, for example, an "init-sec token", a "wrapped token" and a "close token".

2) In addition to authentication, SESAME supports the transmission of the access control privileges of a user. These privileges are carried in a data structure called the PAC (Privileges Attribute Certificate). Privileges supported in SESAME V5 are group-memberships, roles and administration defined local types. In the future support for clearances or capabilities is envisaged. SESAME supports the "push model" where the privileges are pushed towards the target. This allows the principle of least privilege to be supported, where only the privileges that are necessary for performing an operation are presented and disclosed to the target.

3) Privileges are always directly guaranteed by the authority which originally vouched for them. This allows the concept of "direct trust" to be supported because no intermediate security domain is needed to translate the original guaranteed privileges when they are delegated, even across security domains. In practice, this is made possible because the privileges are placed in a data structure that is signed by the issuing domain authority, and thus is directly verifiable.

4) The scheme used to transmit privileges is independent of the scheme used for key management. This allows several key management schemes and their extensions to be supported. In practice, SESAME allows each security domain manager to chose its own key management scheme. Cross-domain relationships can either be based on public key technology or on secret key technology.

5) The control of delegation is a key feature of SESAME. This allows privileges to be transmitted and their use to be restricted to nominated targets or groups of targets.

6) The SESAME GSS-API protocols re-use data structures developed in Kerberos V5 [Kerberos] and SPKM [SPKM] for key management and authentication. SESAME has merged these into a wider framework supporting distributed access control and delegation features.

A more complete overview of SESAME is available in [SESOV].

1.1 Table of Contents

2. THE SESAME TECHNOLOGY

2.1. Overview

The tokens defined in SESAME are intended to be used by application programs according to the GSS API "operational paradigm" (see [RFC-1508] for further details):

> The operational paradigm in which GSS-API operates is as follows. A typical GSS-API caller is itself a communications protocol [or is an application program which uses a communications protocol], calling on GSS-API in order to protect its communications with authentication, integrity, and/or confidentiality security services. A GSS-API caller accepts tokens provided to it by its local GSS-API implementation [i.e., its GSS-API mechanism] and transfers the tokens to a peer on a remote system; that peer

passes the received tokens to its local GSS-API implementation for processing.

Some extensions to the base GSS-API are required in order for applications to take full advantage of the access control and delegation capabilities of SESAME. As these extensions are independent of the SESAME mechanism they are specified in a separate draft [XGSSAPI].

2.2. SESAME Concepts

2.2.1. Access Control Concepts

2.2.1.1. Domains

A `security domain' is a set of elements, a security authority and a set of security relevant activities in which the set of elements are subject to the security policy, administered by the security authority, for the specified activities [ISO 10181-1]. A security domain must support at least, one authentication server (AS), one privilege attribute server (PAS) and may need a key distribution server (KDS).

2.2.1.2. Identities

SESAME supports the three following types of identity:

Authenticated Identity	which is the identity held in the PAS ticket obtained from Authentication Server. This is used to permit the release of Privilege Attributes for use in access control decisions.
Access Identity	which is used in PACs as a Privilege Attribute. This attribute reflects a value given by the PAS administrator. Note that the access identity does not need to be always present within a PAC, because access can be granted using, for example, group-memberships or roles rather than individual identities.
Audit Identity	a value unique to an individual which is used in PACs only for accountability purposes. This is a separate field in the PAC, which reflects a value given by the PAS administrator.

2.2.1.3. Privilege Attributes

Standard Privilege Attribute types are defined so they can be independent of specific end-system representations but which can

be mapped as appropriate in the receiving system.

The privilege attributes which are supported in this specification are :

access identity
primary group
group
role attribute
Domain defined attributes (i.e. defined by the PAS administrator)

Together these provide for the support of a variety of formulations of access control policy, for example Role Based Access Control.

2.2.1.4. Delegation

An initiator may not wish to delegate all his rights, and may want to restrict the area within which the PAC may be used. For that purpose, PACs can be arranged to be valid only for specific nominated targets. A PAC may contain many target names or other target attributes (though in SESAME V5 the only attribute type supported is the Application Trust Group - see next).

Mechanisms are also provided to prevent a PAC from being delegated where this is appropriate.

2.2.1.5. Application Trust Groups (ATGs)

A PAC may contain one or more target or delegate application or "Trust Group" names specifying which targets or delegate-targets the PAC is valid for. A Trust Group name is simply the name of a group of applications, defined by the security administrator, that mutually trust each other not to spoof each others' identities.

In order to allow for a PAC which is usable at all targets a special trust group is defined - the "universal" trust group. A PAC targeted at the universal trust group can still be protected using target controls (as in delegation) which means that such a PAC still cannot be stolen.

2.2.2. Target Access Enforcement Function (targetAEF)

In SESAME the security processing functionality on the target is split between two different entities - the target application and the target access enforcement function (targetAEF)(see ISO IEC 10181-3). This has a number of advantages:

- the number of long-term keys in the system can be reduced since only the targetAEF need share a symmetric key with the security server or possess an asymmetric key pair,

- the security critical code is isolated which makes security evaluation simpler,

- administration is simplified as one targetAEF may "serve" many target applications,

- different administrators can be responsible for the target application and targetAEF,

- as the initiator establishes a key with the targetAEF (see later) this keying information can be re-used whenever another target served by the same AEF is accessed.

A patent from ICL applies to this method (see section 9).

2.2.3. SESAME Key Distribution

There are different key distribution schemes in SESAME. Each depends upon the existence of long term cryptographic keys which held by targets AEFs and KDSs. These keys may be either symmetric or asymmetric. In the case where the keys are symmetric they are always shared between the targetAEF and its KDS.

Initiators may also possess symmetric or asymmetric keys. In the case where an initiator possesses a symmetric key it is a temporary key that will have been established as a result of an earlier authentication.

2.2.3.1. Basic keys and dialogue keys

In SESAME, two separate symmetric keys are established between the initiator and target for the purpose of application data protection. These are known as the integrity and confidentiality dialogue keys.

Another symmetric key, called the basic key, is established between the initiator and targetAEF which is used in PAC protection and to derive the dialogue keys.

The basic key is transmitted from the initiator to the target in a structure called a TargetKeyBlock. The information required to derive the dialogue keys is transmitted in a structure called a DialogueKeyPackage.

2.2.3.2. Basic symmetric key distribution scheme (symmIntradomain)

In this scheme, the initiator shares a temporary secret key with the KDS and the target AEF shares a long term secret key with the same KDS.

To establish a basic key between an initiator and a targetAEF, the initiator KDS returns, as a result of an initiator request, a targetKeyBlock containing a basic key encrypted under the targetAEF's long term secret key. On receipt of the targetKeyBlock,

the targetAEF can extract the basic key directly from it.

An unmodified Kerberos TGS is used as the KDS in this case.

2.2.3.3. Hybrid key distribution scheme (hybridInterdomain)

In this scheme, the initiator shares a temporary secret key with a KDS that is different from the KDS with which the targetAEF shares its long term key. In addition, each KDS possesses an asymmetric key pair.

To establish a basic key between an initiator and a targetAEF, the initiator KDS returns, as a result of an initiator request, a TargetKeyBlock containing a basic key encrypted under a temporary key and the temporary key encrypted under the targetAEF KDS's public key. The TargetKeyBlock is signed using the initiator KDS's private key.

On receipt of the TargetKeyBlock, the targetAEF transmits it to its own KDS, and gets back the basic key encrypted under the long term secret key it (the targetAEF) shares with its KDS.

A modified Kerberos TGS can be used as the KDS in this case.

2.2.3.4. Full public key distribution scheme (asymmetric)

In this scheme, neither the initiator nor the targetAEF uses a KDS. Both the initiator and the targetAEF possesses a private/public key pair.

To establish a basic key with a targetAEF, the initiator constructs a TargetKeyBlock containing a basic key encrypted under the targetAEF's public key. The TargetKeyBlock is signed using the initiator's private key.

On receipt of the TargetKeyBlock, the targetAEF directly establishes a basic key from it.

Modified SPKM code can be used for this scheme.

2.2.3.5. Derivation of Dialogue Keys

Once a basic key has been established between the initiator and targetAEF, the derivation of the dialogue keys can take place. The dialogue keys are derived using key offsetting - see the description of the dialogue key package below.

2.2.4. Use of Cryptography in SESAME

In several countries of the world the use of cryptography is subject to government control, particularly in relation to the hiding of information by enciphering it.

The SESAME architecture has been designed to address these problems. The use of confidentiality is kept to a minimum. It is provided only where it is an essential function (for example in the SESAME key distribution protocols it is necessary to encipher the basic key being distributed). PACs are cryptographically signed, not enciphered. When encipherment of user and system data is a requirement, SESAME allows the algorithms and keys used to be separately specified, permitting them to have characteristics acceptable to the prevailing political environment.

3. GSS-API TOKEN FORMATS

3.1. Token framings

All tokens including context-establishment tokens, per-message tokens, and context-deletion token are enclosed within framing as follows:

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

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

```
{ generic-sesame-mech (v) (y) (z) }
```

Where:

```
generic-sesame-mech ::= OBJECT IDENTIFIER
```

```
iso.org.icd-ecma.technical-report.security-in-open-systems.
authentication-machanism {1.3.12.1.46.1}
```

The value v represents the version of the mechanism. The current version is version 5. The current oid is therefore: {1.3.12.1.46.1.5}

The values of y and z represent architectural options and cryptographic algorithm profiles which are specified in section 5. These values are intended to be negotiated using a generic GSS-API mechanism negotiation scheme like that given in [SNEGO].

The innerContextToken field of context establishment tokens for the SESAME GSS-API mechanism will consist of a SESAME 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 Initial Context 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 context-deletion token for the SESAME GSS-API mechanism will consist of a SESAME token (ContextDeleteToken) containing a tokenId field having the value 01 02 (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.

The innerContextToken field of per-message tokens for the SESAME GSS-API mechanism will consist of a 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.

3.2. InitialContextToken format

```
InitialContextToken ::=  SEQUENCE{
  ictContents    [0]   ICTContents,
  ictSeal        [1]   Seal
}
```

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 targetAEFPart from the initial context token.

```
ICTContents ::=  SEQUENCE {
  tokenId              [0]   INTEGER, -- shall contain X'0100'
  SAId                 [1]   OCTET STRING,
  targetAEFPart        [2]   TargetAEFPart,
  targetAEFPartSeal    [3]   Seal,
  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,
  targetAddress        [9]   HostAddress   OPTIONAL
     -- imported from [Kerberos] and 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 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_ses_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 SESAME mechanism option (see section 6).

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 to 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 are used together to avoid collision between tokens generated by two initiators at the same time. This field enables a simple scheme for replay detection of initial tokens to be supported.

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 SESAME 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 SESAME GSS-API implementation.

```
TargetAEFPart ::=  SEQUENCE {
  pacAndCVs          [0]   SEQUENCE OF CertandECV OPTIONAL,
  targetKeyBlock     [1]   TargetKeyBlock,
  dialogueKeyBlock   [2]   DialogueKeyBlock,
  targetIdentity     [3]   SecurityAttribute,
  flags              [4]   BIT STRING   {
                             delegation         (0)
                         }
}
```

Note 1: Individual PACs have validity of their own, thus it is not sensible to have an overall separately specified validity period for the whole context.

pacAndCVs
The initiator's privileges and security attributes to be used for this Security Association. This field is not present when the association does not require initiator privileges or security attributes. This field contains the PAC together with associated PAC protection information. In this specification exactly one of these should be present.

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

dialogueKeyBlock
A dialogue key block used by the targetAEF 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 targetAEF to validate the PAC. Can also be used by the targetAEF 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.

The Seal structure is used in this Token and other tokens.

```
Seal ::=  SEQUENCE{
  sealValue          [0]   BIT STRING,
  symmetricAlgId     [1]   AlgorithmIdentifier        OPTIONAL,
  hashAlgId          [2]   AlgorithmIdentifier        OPTIONAL,
  targetName         [3]   SecurityAttribute          OPTIONAL,
  keyId              [4]   INTEGER                    OPTIONAL
}
```

sealValue
The value of the seal. It is the result of a symmetric encryption of a hash value of a set of octets (which are the DER encoding of some ASN.1 type)

symmetricAlgId
An optional indicator of the sealing algorithm.

hashAlgId
Only present if the symmetricAlgId does not specify which hashing algorithm is used.

targetName
This field identifies the targetAEF or target with which the symmetric key used for the seal is shared.

keyId
This serial number together with the targetName uniquely identifies the symmetric key used in the seal.

3.3. TargetResultToken

This token is returned by the target if the mutual-req flag is set in the Initial Context Token. It serves to authenticate the target to the initiator, since only the genuine target 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,
}
```

trtContents

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

seq-number
When present, specifies the target'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 above.

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 targetAEFPart from the initial context token.

3.4. ErrorToken

```
ErrorToken ::= SEQUENCE {
   tokenType     [0]   OCTET STRING VALUE X'0300',
   etContents    [1]   ErrorArgument,
}
```

etContents
Contains the reason for the creation of the error token. The different reasons are given as minor status return values.

```
ErrorArgument ::=  ENUMERATED {
  gss_ses_s_sg_server_sec_assoc_open              (1),
  gss_ses_s_sg_incomp_cert_syntax                 (2),
  gss_ses_s_sg_bad_cert_attributes                (3),
  gss_ses_s_sg_inval_time_for_attrib              (4),
  gss_ses_s_sg_pac_restrictions_prob              (5),
  gss_ses_s_sg_issuer_problem                     (6),
  gss_ses_s_sg_cert_time_too_early                (7),
  gss_ses_s_sg_cert_time_expired                  (8),
  gss_ses_s_sg_invalid_cert_prot                  (9),
  gss_ses_s_sg_revoked_cert                       (10),
  gss_ses_s_sg_key_constr_not_supp                (11),
  gss_ses_s_sg_init_kd_server_ unknown            (12),
  gss_ses_s_sg_init_unknown                       (13),
  gss_ses_s_sg_alg_problem_in_dialogue_key_block  (14),
  gss_ses_s_sg_no_basic_key_for_dialogue_key_block (15),
  gss_ses_s_sg_key_distrib_prob                   (16),
  gss_ses_s_sg_invalid_user_cert_in_key_block     (17),
  gss_ses_s_sg_unspecified                        (18),
  gss_ses_s_g_unavail_qop                         (19),
  gss_ses_s_sg_invalid_token_format               (20)
}
```

3.5. Per Message Token formats

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
        -- seal over the pmtContents being protected
   }

   PMTContents ::=  SEQUENCE {
     tokenId              [0]   INTEGER, -- shall contain X'0101'
                                         -- for a MIC token and
                                         -- X'0201' for a Wrap token.

     SAId                 [1]   OCTET STRING,
     seq-number           [2]   INTEGER
   OPTIONAL,
     userData             [3]   CHOICE {
                                plaintext      OCTET STRING,
                                ciphertext     OCTET STRING
                         }                             OPTIONAL,
     directionIndicator   [4]   BOOLEAN                OPTIONAL
   }
```

pmtContents

tokenId
SAId
See above for a description of these fields

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 target 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
FALSE indicates that the sender is the context initiator,
TRUE that the sender is the target.

pmtSeal
See specific token type narratives below.

3.5.1. 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 above. Fields specific to the MICToken are:

userData
Not present for MIC Tokens.

pmtSeal
The Checksum is calculated over the DER encoding of the pmtContents field with the user data temporarily placed in the userData field. The userData field is not transmitted.

3.5.2. 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 above. Fields specific to the WrapToken are:

userData

Present either in plain text form (the choice is plaintext), or encrypted (choice ciphertext). If the data is encrypted, it is

performed using the Confidentiality Dialogue Key, and as in [Kerberos], an 8-byte random confounder is first prepended to the data to be encrypted.

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

3.6 ContextDeleteToken format

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 above.

cdtSeal
See above for a general description of the use of this construct.

4. DATA ELEMENT DEFINITIONS

In this section we give the details of the structures which are present in the tokens defined above.

These ASN.1 definitions represent the profile of the ASN.1 types

defined in [ECMA-219] which are implemented in the SESAME project. In some cases CHOICEs and OPTIONAL fields which are defined by ECMA have been omitted as they are not supported in this version of SESAME. In order to retain compatibility this leads to a non-obvious numbering for tags.

In this specification all of the type specifying object identifiers are below the arc:

```
generic-sesame-mech ::=  OBJECT IDENTIFIER {1.3.12.1.46}
   -- top of the SESAME types arc
```

4.1. Access Control Data Elements

The ASN.1 definitions for the PAC and related data structures. The full ASN.1 specification can be found in [ECMA-219] and in Annex A.

4.1.1. Generalised certificate (GeneralisedCertificate)

A Generalised Certificate (PAC) consists of a certificateBody and checkValue, the latter containing a digital signature applied to the former. The CertificateBody is formed of two parts: a commonContents and a specificContents part.

The "commonContents" fields collectively serve to provide generally required management and control over the use of the PAC.

The "specificContents" fields are different for different types of certificate, and contain a type identifier to indicate the type. In this specification only one type is defined: the Privilege Attribute Certificate (PAC).

The "checkValue" fields are used to guarantee the origin of the certificate.

The next sections describe these three main structural components of the Generalised Certificate.

4.1.1.1. Common Contents fields (CommonContents)

The common contents field of the PAC is made of the following fields:

comConSyntaxVersion
Identifies the version of the syntax of the combination of the commonContents and the checkValue fields parts of the certificate.

issuerDomain

The security domain of the issuing authority. Not required if the form of issuerIdentity is a full distinguished name, but required

if other forms of naming are in use, such as proprietary identifiers.

issuerIdentity
The identity of the issuing authority for the certificate.

serialNumber
The serial number of the certificate (PAC) as allocated by the issuing authority.

creationTime
The UTC time that the certificate was created, according to the authority that created it.

validity
A pair of start and end times within which the certificate is deemed to be valid.

algId
The identifier of the symmetric or of the asymmetric cryptographic algorithm used to seal or to sign the certificate. If there is a single identifier for both the encryption algorithm and the hash function, it appears in this field.

hashAlgId
The identifier of the hash algorithm used in the seal or in the signature.

4.1.1.2. PAC Specific Certificate Contents (PACSpecificContents)

The specific contents field of the PAC is made of the following fields:

The pacSyntaxVersion is defaulted.

protectionMethods
A sequence of optional groups of Method fields used to protect the certificate from being stolen or misused. For a full description see section 4.1.3.

The pacType is defaulted.

privileges
Privilege Attributes of the principal in the form of security attributes. For a full description ofsecurity attributes see section 4.1.2.

restrictions
not supported in SESAME V5 - see [ECMA-219] for a full description.

miscellaneousAtts

Attributes that are neither privilege attributes nor restrictions. Audit identity is one such attribute.

TimePeriods
further time period restriction over and above that specified in commonContents.

4.1.1.3. Check value (CheckValue)

In this specification a PAC is protected by being digitally signed by the issuer.

A signature may be accompanied by information identifying the Certification Authority under which the signature can be verified, and with an optional convenient reference to or the actual value of the user certificate for the private key that the signing authority used to sign the certificate.

A signature is composed of the following fields:

signatureValue
The value of the signature. It is the result of an asymmetric encryption of a hash value of the certificateBody.

issuerCAName
The identity of the Certification Authority that has signed the user certificate corresponding to the private key used to sign this certificate.

caCertInformation
Contains either just a certificate serial number which together with the issuerCAName uniquely identifies the user certificate corresponding to the private key used to sign this certificate, or a full specification of a certification path via which the validity of the signature can be verified. The latter option follows the approach used in [ISO/IEC 9594-8].

4.1.2. Security Attributes (SecurityAttribute)

The security attribute is a basic construct made up of :

attributeType
Defines the type of the attribute. Attributes of the same type have the same semantics when used in Access Decision Functions, though they may have different defining authorities.

definingAuthority
Indicates the authority responsible for defining the value within the attribute type. Some policies demand that multiple sources of values for a given attribute type be supported (e.g. a policy accepting attribute values defined outside the security domain), These policies give rise to a risk of value clashes. The definingAuthority field is used to separate these values. When not

present, the value defaults to the name of the authority that issued the certificate containing the attribute.

securityValue
The value of the security attribute. Its syntax is can be either one of the basic syntaxes for attributes or a more complex one determined by the attribute type.

4.1.3. Protection Methods

Protection methods are grouped in methodGroups. See section 4.1.3.5 for the significance of these groups. Protection methods are formed as a combination of methodId and methodParams. The methodParams are formed as a sequence of Mparm constructs. Each methodId determines a syntax for Mparm.

methodId
Identifies a protection method. Methods can be used in any combination, and except where stated otherwise, multiple occurrences of the same method are permitted. The choice of methodId determines the permitted choices of method parameters in the methodParams construct .

methodParams
Parameters for a protection method formed as a sequence of individual method parameter constructs (Mparm).

There are four basic protection methods, as described below.

4.1.3.1. "Control/Protection Values" protection method

This is known as the CV/PV method. A patent from Bull applies to this method (see section 9).

This method allows a PAC to be used by proxy while at the same time preventing it from being stolen by an eavesdropper. The PAC can be forwarded to any target or group of targets. The owner of the PAC need not know in advance. But if this information is known, use of the PAC can be confined to any nominated target or group of targets. Ownership of a PAC can be proven by presenting a CV value which matches a public value contained in the PAC. The two values must relate through the relationship: PV = OWF (CV), where OWF is a one way collision resistant hash function. Unless otherwise specified, the one-way function that is used for this method is MD5.

The MethodId for this is: controlProtectionValues

The syntax choice of Mparm for this method is: pValue.

A maximum of one PV method is permitted in each method group. More than one method group can be specified, each containing a PV.

Associated with each PV is a certificate Control Value (CV)

external to the certificate. The CVs are carried encrypted in the ECV construct.

When the client sends a PAC to a targetAEF, one or more CVs are also sent encrypted under the basic key used to communicate with the target AEF. The targetAEF knows the one-way function and therefore is able to verify that the client knows a CV which corresponds to a PV in the certificate. If the other controls in the PV's method group are passed, the PAC is acceptable under this method group.

The targetAEF now knows the value of the CVs that have been sent to it, and can make available one or more of the values to the infrastructure supporting that application for forwarding with the certificate to other applications. By including target qualification controls in the method group, proxy can be confined to nominated target applications. One use for this might be, for example, all of the application servers in a distributed service.

4.1.3.2. "Primary Principal Qualification" protection method

This is known as the PPID (Primary Principal IDentifier) method. A patent from ICL applies to this method.

The MethodId for this is: ppQualification

This method protects the certificate from being stolen, by confining its use to be from one or more nominated "Primary Principals" as defined in [ECMA-219]. In its most restrictive form it permits a certificate to be used only from the Primary Principal of the client entity to which it was originally issued, preventing delegation of the certificate. However it can also be used to permit delegation, when the required attributes of the proxy application are precisely known in advance. The method by itself does not limit the number of target applications at which a PAC might be accepted. However, by including separate target qualification controls in the method group, delegation can be confined to nominated target applications.

The syntax choice of Mparm for this method is : securityAttribute

A sequence of Mparm constructs is permissible, resulting in multiple nominated Primary Principals being capable of being permitted.

At least one of these attributes must be possessed by any Primary Principal from which this certificate is to be validly used if the PAC is to be accepted under this method. When a targetAEF receives such a certificate, it is responsible for comparing these attributes with attributes placed within the targetKeyBlock construct and associated with the initiating Primary Principal.

When a symmetric key distribution scheme is in use the attribute values are placed in the target key block by the trusted server

(the KDS) which created it. If there is no KDS, as in the case of pure asymmetric key distribution, they are present in the public key certificate of the initiator that is sent with the PAC.

The attribute value used here is termed the primary principal identifier (PPID) and takes one of two forms depending on the key distribution scheme used by the initiator. For the symmetric intradomain and hybrid interdomain schemes the PPID takes the form of a random bit string which is also sent in the authorization data field of the Kerberos ticket. For the asymmetric scheme the PPID is constructed from the certificate serial number and the CA name for the initiator's X.509 public key certificate.

4.1.3.3. "Target Qualification" protection method

The MethodId for this is: targetQualification.

This method protects the PAC from misuse by allowing its use only by one or more nominated target applications and at the same time instructing the target AEF to prevent it from being forwarded.

The syntax choice of Mparm for this method is : securityAttribute

A sequence of Mparm constructs is permissible, resulting in multiple security attributes being present.

Target AEFs receiving such PACs will compare the values found in the PAC with the attributes of the target application. If the target application possesses one of the attributes in one of the occurrences of this method that is present in a method group, the Certificate is deemed to be acceptable under this method in this group but the target AEF is expected not to allow the use of the certificate for access to further target applications.

4.1.3.4. "Delegate/Target Qualification" protection method

The MethodId for this is: delegateTargetQualification

This method protects the PAC from misuse by confining its validity to one or more nominated target applications and at the same time instructing the target AEF to allow the PAC to be forwarded.

The syntax choice of Mparm for this method is: securityAttribute

A sequence of Mparm constructs is permissible, resulting in multiple security attributes being present.

The target AEF makes the comparisons described in the previous section, but if the checks are passed, the nominated target application(s) is/are acceptable as both an accessible target and

as a delegate. The attributes carried by the certificate are valid at the target application(s) for authentication or access control

purposes and the target AEF will allow the PAC to be forwarded to further target applications.

4.1.3.5. Combining the methods

The general rule for validating a PAC when received by a target AEF is as follows:

If the PAC is properly signed by an authority recognised by this targetAEF and is within the valid time periods then the protection methods in each group are tested in turn as described below until one of the groups is passed or the PAC is declared invalid.

A method group is passed if it is empty, or if:

> the PAC is for the nominated target application under any target or delegateTarget qualification method in this group
>
> and
>
> tests on any ppQualification or controlProtectionValues method in the group succeed. If more than one of these is present, at least one must be passed. If only one is present it must be passed. If none of them is present this last check is not required.

If the group that has been passed only validates the certificate for its recipient being a target, then further groups are checked to see if the recipient is also valid as a delegate/target. If, following these additional checks, a recipient is still valid only as a target, the targetAEF is responsible for preventing its use for access to further target applications.

4.1.4. External Control Values Construct (ECV)

Whenever the protection controlProtectionValues method is in place, when a PAC protected under that method is being presented as authorisation for an operation, it may be accompanied by one or more control values and indices to the method occurrences in the certificate to which they apply.

crypAlgIdentifier
This specifies the encryption algorithm of the control values.

cValues
An ECV construct contains in the encryptedCvalueList field a list of control values encrypted under the basic key protecting the operation. The whole list is encrypted in bulk, but the in-clear contents of this field are expected to have the syntax CValues.

The CValues is composed of a sequence of tuples, each one being

composed of an index of the method occurrence in the certificate, starting at 1, and the value of the CV.

4.2. Basic Key Distribution

The TargetKeyBlock is structured as follows:

- an identifier (kdSchemeOID) for the key distribution scheme being used, which takes the form of an OBJECT IDENTIFIER,

- a part which, if present, the target AEF needs to pass on to its KDS (targetKDSPart - will be present only when the target AEF's KDS is different from the initiator's),

- a part which, if present, can be used directly by the target AEF (targetPart).

When a targetAEF using a separate KDS receives the targetKeyBlock, it first checks whether it supports the key distribution scheme indicated in kdsSchemeOID. Two different cases need to be considered:

1) Only the targetPart is present. The target AEF computes the basic key directly, using the information present in the TargetPart. The syntax of targetPart is scheme dependent. Expiry information can optionally be present in targetPart. If supported by the scheme, the Primary Principal attributes of the initiator will also be present for PAC protection under the Primary Principal Qualification method (see above).

2) Only the targetKDSPart is present. The targetAEF forwards TargetKeyBlock to its KDS. In return it receives a scheme dependent data structure which by itself allows the target AEF to determine the basic key and, if supported by the scheme, the Primary Principal attributes of the initiator for PAC protection purposes. Expiry information can optionally be present in the targetKDSPart.

The form of this information depends on the key distribution configuration in place.

The TargetKeyBlock is composed of the following fields:

kdSchemeOID
Identifies the key distribution scheme used. Allows the targetAEF to determine rapidly whether or not the scheme is supported. It also allows for the easy addition of future schemes.

targetKDSpart

Part of the Target Key Block which is processable only by the KDS of the target AEF. This part is sent by the target AEF to its

local KDS, in order to get the basic key which is in it. It must always contain the name of a target "served" by the targetAEF in question. The mapping between the name of the application and the name of the target AEF is known to the target AEF's KDS which is able to authenticate which target AEF is issuing the request for translating the targetKDSpart. It can then verify that the AEF is one which is responsible for the application name contained in the targetKDSpart. If it is, the key is released and is sent protected back to the requesting AEF. The targetKDSpart includes data that enables the KDS of the target AEF to authenticate the KDS of the initiator. When the "Primary Principal Qualification" protection method needs to be used for the PAC, unless there is an accompanying targetPart, targetKDSpart contains the appropriate primary principal security attributes.

targetPart
Part of the Target Key Block which is processed only by the target AEF. When there is no targetKDSpart it is processable directly; otherwise it can only be processed after the targetKDSpart has been processed by the KDS of the target AEF, and the appropriate Keying Information has been returned to the AEF. The targetPart construct should include data that enables the target AEF to authenticate the KDS of the initiator. When the "Primary Principal Qualification" protection method needs to be used for the PAC, targetPart must contain the primary principal security attributes.

The following table shows the different syntaxes used for targetKDSpart and targetPart for the 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
hybridInterdomain	{kd-schemes 3}	PublicTicket	Missing
asymmetric	{kd-schemes 6}	Missing	SPKM_REQ

Table 1 - Key Distribution Scheme OBJECT IDENTIFIERs

The syntax of PublicTicket is given in appendix A.1, and the syntax of Ticket (copied from [Kerberos]) is given in appendix A.2. The syntax of SPKM_REQ (copied from [SPKM])is given in appendix A.3.

The OBJECT IDENTIFIERs that are for use in the kdSchemeOID field of TargetKeyBlock are formally derived from the kd-schemes OBJECT IDENTIFIER

The SPKM_REQ construct used in scheme 6 requires a sequence of key establishment algorithm identifier values to be inserted into the

key_estb_set field. The OBJECT IDENTIFIER sesame-key-estb-alg is defined as the (single) key establishment "algorithm" for the SESAME mechanism.

kd-schemes
This OBJECT IDENTIFIER is the top of the arc of key distribution scheme OBJECT IDENTIFIERs defined in this specification.

symmIntradomain
This OBJECT IDENTIFIER indicates the basic symmetric key distribution scheme described in section 2.2.3.2. 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 appendix A.2). The profile of the ticket that is supported this scheme can be found in Table 2.

hybridInterdomain
This OBJECT IDENTIFIER indicates the hybrid scheme described in section 2.2.3.3. The targetKDSpart contains a PublicTicket (defined in section 4.2.2). The targetPart field is not supplied. The PublicTicket contains a Kerberos Ticket. The profile supported in this scheme can be found in Table 3.

asymmetric
This OBJECT IDENTIFIER indicates the scheme described in section 2.2.3.4. The targetKDSpart is not supplied and the targetPart contains an SPKM_REQ. The syntax of SPKM_REQ is given in appendix A.3. The profile of SPKM_REQ that is supported in this scheme is given in Table 4.

sesame-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 SESAME.

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.

4.2.1. Data elements for the Symmetric Intradomain kd-scheme

The full ASN.1 for the Kerberos elements used by the SESAME GSS-API mechanism is given in appendix A.2. This section specifies the specific contents of the Kerberos Ticket's authorization_data field required by the SESAME GSS-API mechanism.

Essentially this construct (SESAME-AUTHORIZATION-DATA) contains the PPID of the context initiator.

ppidType
Indicates the type of authorisation data as being SESAME authorisation data.

ppidValue
This value is used in the ppQualification PAC protection method as defined in section 4.1.3.2.

4.2.2. Data elements for the Hybrid interdomain kd-scheme.

The PublicTicket contains the following fields:

krb5Ticket
The Kerberos Ticket which contains the basic key. The encrypted part of this ticket is 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).

signedPKBPart
The part of the publicKeyBlock which is signed. The keyEstablishmentData field contains the KeyEstablishementData (defined in section 4.2.4), i.e. the actual encrypted temporary key (see section 2.2.3). The encryptionMethod indicates the algorithm used to encrypt the encryptedKey. The issuingKDS is the name of the KD-Server who produced the PublicTicket. The uniqueNumber is a value (containing a timestamp and a random number) 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.

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

certificate
If present, contains the public key certificate of the issuing KDS.

The KeyEstablishmentData contains the following fields :

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 SESAME KD-schemes.

nameHashingAlg
Specifies the algorithm which is used to calculate the hashedName

field of the PlainKey.

hniPlainKey
hniIssuingKDS
Used as input to a hashing algorithm as a general means to prevent ciphertext stealing attacks.

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

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

targetName
If present, contains the name of the target for which the PublicTicket was originally produced. This may be different from the targetIdentity field of the initialContextToken if caching of PublicTickets has been implemented.

4.2.3. Data elements for the asymmetric kd-scheme

The targetPart contains an SPKM_REQ. The syntax of SPKM_REQ is given in appendix A.3. The profile of SPKM_REQ that is supported in this scheme is given in Table 4.

4.3. Dialogue Key Block

Dialogue Key Block constructs are used to specify how the integrity dialogue key and confidentiality dialogue key should be derived from the basic key, and specify the cryptographic algorithms with which the keys should be used.

The DialogueKeyBlock is composed of the following fields:

integKeySeed
A random number, optionally concatenated with a time value to ensure uniqueness, used as input to the one way function specified in integKeyDerivationInfo.

confKeySeed
A random number, optionally concatenated with a time value to ensure uniqueness, used as input to the one way function specified in confKeyDerivationInfo.

integKeyDerivationInfo
Key derivation information for the integrity dialogue key, as

follows:

owfId
The one way algorithm which takes the basic key XOR the seed as input, resulting in the integrity dialogue key.

keySize
The size of the key in bits. If the algorithm identified by owfId produces a larger key, it is reduced by masking to this length, losing its most significant end.

confKeyDerivationInfo
Key derivation information for the confidentiality dialogue key. The fields in this construct have the same meanings as defined above for the integrity dialogue key.

Note:
It may be insecure to specify the same derivation algorithms and seeds for both integrity and confidentiality dialogue keys, particularly if they are to be of different lengths.

integDKuseInfo
Information describing how the integrity dialogue key is to be used, as follows:

useAlgId
The symmetric or asymmetric reversible encryption algorithm with which the integrity dialogue key is to be used.

useHashAlgId
The one way function with which the integrity dialogue key is to be used. It is the hash produced by this algorithm on the data to be protected which is encrypted using useAlgId.

confDKuseInfo
Information describing how the confidentiality key is to be used. The useHashAlgId construct is not used here.

4.4. Attribute Definitions

4.4.1. Privilege attributes

4.4.1.1. Access Identity

The access identity represents an identity that the principal is permitted to use for access control purposes.

4.4.1.2. Group

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

4.4.1.3. 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.

4.4.1.4. Role attribute

The role attribute represents a principal's role as might be used in a role based access control policy. For example it can represent a job position an individual may have within a company.

4.4.2. Miscellaneous attributes

4.4.2.1. Audit Identity

Audit identity represents an identity unique to principal to be used for accountability purposes.

4.4.2.2 Other

Other miscellaneous attributes are
defined in ECMA-219 but are not currently supported in SESAME V5.

4.4.3. Qualifier Attributes

When a targetQualifiication or delegateTargetQualification method is present in the PAC, the syntax used for the method parameters is securityAttribute.

4.4.3.1. Target Attributes

Within a PAC protection method, targets can be identified by name or other attributes to indicate whether they are allowed to accept or both accept and forward that PAC.

Other than name, the only target attribute supported in SESAME V5 is the Application Trust Group (see below)..

4.4.3.2. Application Trust Groups

Within a PAC protection method, an application trust group name specifies the name of a set of targets allowed to accept or both accept and forward that PAC.

The universal application trust group (see section 2) is specified by using an empty string value. See also section 2.2.1.5.

5. ALGORITHMS AND CIPHERTEXT FORMATS

Cryptographic and hashing algorithms are used for various

purposes within the SESAME GSS-API mechanism. This section 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.

The table below summarises the different uses to which algorithms are put within the SESAME GSS-API mechanism.

Use Reference	Description of use	Type of Algorithm
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 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	key estab. within SPKM_REQ	(fixed value)
11	confidentiality dialogue key derivation	OWF
12	confidentiality dialogue key use	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 3, 5, 7
Class 2: all OWFs:
Uses 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: Data confidentiality, symmetric:
Use 12

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

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 5: Defaulted
Class 1	DES/64	DES/64	RC4/128	separately agreed default
Class 2	MD5	MD5	MD5	separately agreed default
Class 3	RSA	RSA	RSA	separately agreed default
Classes 4 and 5	RSA	RSA	RSA	separately agreed default
Class 6	DES/64	None	RC4/40	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 exportable under many countries' legislations,

- 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.

The next section shows how these algorithm profiles can be used

to extend the architectural key distribution schemes to form negotiable SESAME mechanism choices.

6. SESAME MECHANISM NEGOTIATION

Preceding sections have separately defined the alternatives allowed by the generic SESAME mechanism in terms of key distribution schemes and the use of cryptographic and hash algorithms within the data elements.

This section brings these together by defining the specific SESAME 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 (like [SNEGO]).

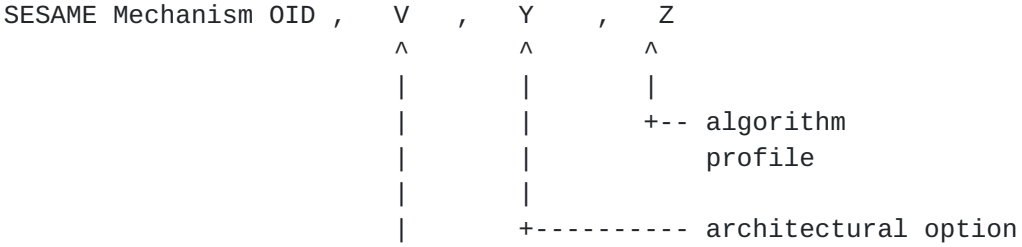
The approach is to use the key distribution schemes to form broad architectural mechanism options, as follows (more options are defined by ECMA, hence the numbering):

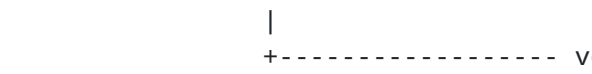
Architectural Mechanism Number	Description of Mechanism Option	Key Distribution Scheme(s)
2	Symmetric key distribution	symmIntradomain
3	Symmetric initiator and target; Asymmetric KD-Servers;	symmIntradomain; hybridInterdomain
6	Asymmetric Initiator and Target	asymmetric

Table 7 - Key Distribution Mechanism Options

Each of the security mechanism options described above represents a key distribution scheme.

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




version

Thus a SESAME V5 mechanism using a fully symmetric key distribution scheme and an exportable cryptographic algorithm profile would have an OBJECT IDENTIFIER of:

{ generic-sesame-mech (5) (2) (3) }

A SESAME 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-sesame-mech (5) (6) (2) }

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

Where information is returned from the target to the initiator as a result of negotiation then for the SESAME mechanism the information should contain the public key certificates required for the initiator to be able to use the selected KD-Scheme. For example, if the asymmetric KD-Scheme is to be used the target should return to the initiator the public key certificate of the targetAEF (containing the target's own name in the extensions field). The syntax of the mechanism specific information is the `Certificates' ASN.1 type defined in the AuthenticationFramework. (To allow multiple certificates to be passed to the initiator.)

7. NAME TYPES

Because [Kerberos] does not support Directory Names (DNs), SESAME uses two distinct naming conventions, Kerberos and X.500.

7.1. Kerberos naming

SESAME uses the Kerberos V5 Authentication Server protocol for password based authentication, so SESAME principals are given Kerberos principal names. Moreover, the SESAME security domain is equivalent to a Kerberos realm, so Kerberos realm names are used to identify SESAME security domains. In SESAME, an entity that uses the normal Kerberos V5 authentication via a password is given a printable Kerberos principal name of the form :

<principal_name>@<realm_name>

Notes:
1. Components of a name can be separated by `/`.
2. The separator `@` signifies that the remainder of the string following the `@` is to be interpreted as a realm identifier. If no `@` is encountered, the name is interpreted in the context of

the local realm. Once an `@` is encountered, a non-null realm name, with no embedded `/` separators, must follow. The `\`

character is used to quote the immediately-following character.

SESAME reserves two specific Kerberos principal names for its own use:

- for the SESAME Security Server (containing the AS, PAS and KDS):

  krbtgt/<realm_name>@<realm_name>

- for the SESAME PVF :

  pvf/<host_name>/<realm_name>@<realm_name>

The realm_name in each of these constructs is repeated for compatibility with Kerberos.

Note that a <host_name> or a <realm_name> might take the form of an Internet Protocol domain name, and so a name like:

  pvf/mybox.bull.fr/sesame.bull.fr@sesame.bull.fr

is a valid principal name for a SESAME PVF.

When invoking gss_import_name, a Kerberos principal name type can be identified using either gss_ses_krb5_oid or GSS_KRB5_NT_PRINCIPAL_NAME symbolic names. A Kerberos service name type can be identified using either gss_ses_krb5_oid or GSS_KRB5_NT_HOSTBASED_ SERVICE_NAME symbolic names.

7.2. Directory Naming

As described elsewhere, SESAME uses public key technology supported by Directory Certificates, so for this purpose SESAME entities are given DNs. Such names are built from components separated by a semicolon. The standardised keywords supported by SESAME are :

```
CN (common-name).
S  (surname),
OU (organisational-unit),
O  (organisation),
C  (country),
```

So an example of a DN supported at SESAME is:

  CN=Martin;OU=sesame;O=bull;C=fr

SESAME defines a set of reserved common-name parts for DNs for the core SESAME security components, as follows:

the PAS: CN=SesamePAS.<realm_name>[;...]
the AS: CN=SesameAS.<realm_name>[;...]

the KDS:	CN=SesameKDS.<realm_name>[;...]
the PVF:	CN=pvf.<host_name>[;...]
the security domain:	CN=SesameDomain.<realm_name>[;...] where <realm_name> is the name of the Kerberos realm to which the entity belongs.

Note that there is no generic rule for mapping the Directory Name of a SESAME entity to its Kerberos principal name, so SESAME provides an explicit mapping in a principal's Directory Certificate, using the extensions field of the extended Directory Certificate syntax (version 3) to carry the principal's Kerberos name.

Note also that in the case of a PVF's Directory Certificate, the names of the applications supported by the PVF are also held in this field, preceded by the Kerberos principal name of the PVF itself. In the absence of such a certificate (i.e. if the PVF does not have a key pair of its own) the list of application names can be held (e.g. in a file) in the KDS.

In SESAME the syntax of the Login Name is imported from the Kerberos Version 5 GSS-API Mechanism. This form of name is referred to using the symbolic name: GSS_KRB5_NT_PRINCIPAL. Syntax details are given in [KRB5GSS].When a principal possesses a private key for authentication, the login name is also stored in an extension field of the principal's Directory Certificate so that it can be linked to the principal's Distinguished Name.

8. SECURITY CONSIDERATIONS

Security issues are discussed throughout this memo.

9. PATENTS

Three patents apply. One from Bull and two from ICL.

9.1. BULL PATENT

A patent with the French number 2,662,007 and the French title "Procedure d'obtention d'une attestation en clair securisee dans un environnement de systeme informatique distribue " (Method for obtaining a securitized clear text attestation in a distributed data processing system environment) has been filed on May 10, 1990 under the number 90.05829 and is also registered in the following countries under the following numbers :

- European Patent No 91401138.2 (designated states: Germany, France, GB, Italy, Netherlands, and Sweden),
- Canadian Patent No 2,041,761,
- US Patent No 5,214,700.

The inventors are : Philippe Caille and Denis Pinkas.

9.2. ICL PATENTS

9.2.1. PAC USE MONITOR (C1167)

A patent based on GB 9010603.0 with the title "Access Control in a Distributed Computer System" has been filed on May 11, 1990 and has also be registered in the following countries under the following numbers :

- Australia Patent No: 634653 granted: 25/02/93
- European Application No: 91303752.9 filed: 25/04/91
  (Designated states: Germany, France, GB, Italy, Netherlands.)
- United States Patent No: 5339403 granted: 16/08/94
- South Africa Patent No: 91/3322 granted: 12/12/91

The inventor is : Parker T A.

It uses the term "PAC use monitor" which corresponds to what is called in this specification the "targetAEF".

9.2.2. PROXY CONTROL (C1179)

A patent based on GB 9104909.8 with the title "Access Control in a Distributed Computer System" has been filed on August 3, 1991 and has also be registered in the following countries under the following numbers :

- Australia Patent No: 655960 granted: 19/01/95
- European Application No:92301081.3 filed: 19/02/92
  (Designated states: Belgium, Germany, France, GB, Italy)
- Japan Application No 48618/1992 filed: 05/03/92
- United States Patent No: 5220603 granted: 15/06/93
- South Africa Patent No: 92/1425 granted: 15/09/92

The inventor is : Parker T A.

It uses the term "Proxy control" which corresponds to what is called in this specification the PPID method.

10. ACKNOWLEDGEMENTS

The SESAME project has been carried out by Bull SA, ICL and Siemens (SNI, Siemens ZFE and SSE) under part funding from the CEC as RACE project R2051.

With apologies to those omitted, the following are amongst the people who have made significant contributions to the ECMA and SESAME work: Helmut Baumgaertner, John Cosgrove, Philippe Caille, Hiep Doan, Belinda Fairthorne, Peter Hartmann, Keith Howker, Per Kaijser, Jacques Lebastard, Ronan Long, Piers McMahon,

Frank O'Dwyer, Denis Pinkas, Mike Roe, Laurent Rouilhac, Jean Louis Roule, Don Salmon, Asmund Skomedal and Mark Van DenWauver.

11. REFERENCES

ECMA-219 ECMA-219 Second Edition, March 1996, Authentication and Privilege Attribute Application with related key distribution functions. This standard is available free of charge from: ECMA 114 Rue du Rhone CH-1204 Geneva (Switzerland).
Internet : helpdesk@ecma.ch
This standard can also be downloaded using one of the following URLs:
ftp://ftp.ecma.ch/ecma-st/e219-doc.exe ;
ftp://ftp.ecma.ch/ecma-st/e219-exp.txt ;
ftp://ftp.ecma.ch/ecma-st/e219-pdf.pdf ;
ftp://ftp.ecma.ch/ecma-st/e219-psc.exe .

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/IEC 9594-8 ISO/IEC 9594-8, Information Processing Systems - Open Systems Interconnection - The Directory - Part 8: Authentication Framework (X.509)

KERB5GSS Internet RFC 1964, The Kerberos Version 5 GSS-API Mechanism (J. Linn, June 1996)

XGSSAPI draft-ietf-cat-xgssapi-acc-cntrl-01.txt: Extended Generic Security Service APIs: XGSS-APIs (Denis Pinkas and Piers McMahon, July 1996)

SESOV SESAME Overview, Version 4, (Tom Parker and Denis Pinkas, December 1995)

SPKM RFC 2025 The Simple Public-Key GSS-API Mechanism (C. Adams, OCtober 1996)

SNEGO draft-ietf-cat-snego-01 Simple GSS-API Negotiation Mechanism (Eric Baize and Denis Pinkas, October 1996)

12. AUTHOR'S ADDRESSES

Eric Baize,
Bull HN - MA02/211S
Technology Park
Billerica, MA 01821,
USA.

email: E.Baize@ma02.bull.com

Stephen Farrell
Software and Systems Engineering Ltd.
Fitzwilliam Court,
Dublin 2,
IRELAND.

email: Stephen.Farrell@sse.ie

Tom Parker,
The Solution Centre,
ICL,
Lovelace Road,
Bracknell,
Berkshire RG12 8SN
UK

email: t.a.parker@win0199.wins.icl.co.uk

APPENDIX A: ASN.1 MODULE DEFINITIONS

A.1. SESAME ASN.1 Definitions

```
   SESAME-gss-api-types  { tbs }

   DEFINITIONS ::=

   BEGIN

   -- exports everything

   IMPORTS

     Name
      FROM  InformationFramework
            {joint-iso-ccitt(2) ds(5) module(1)
            informationFramework(1) }

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

     HostAddress, Ticket
      FROM SESAME-Kerberos-Definitions { tbs }

     SPKM-REQ
      FROM SESAME-SPKM-Definitions { tbs };

   -- OBJECT IDENTIFIERS

   access-identity-privilege ::=  OBJECT IDENTIFIER
        { privilege-attribute 2 }

   audit-id-misc ::=  OBJECT IDENTIFIER { misc-attribute 2 }

   generic-sesame-mech ::= OBJECT IDENTIFIER {1.3.12.1.46.1}

   generic-sesame-oids ::=  OBJECT IDENTIFIER {1.3.12.1.46}
      -- top of the SESAME types arc

   group-privilege ::=  OBJECT IDENTIFIER { privilege-attribute 4 }

   kd-schemes   OBJECT IDENTIFIER ::=  { generic-sesame-oids 9}
      -- ECMA-defined arc for SESAME key distribution schemes
   symmIntradomain        OBJECT IDENTIFIER ::=  {kd-schemes 1}
   hybridInterdomain      OBJECT IDENTIFIER ::=  {kd-schemes 3}
   asymmetric             OBJECT IDENTIFIER ::=  {kd-schemes 6}
```

```
      -- supported key distribution schemes
```

```
   misc-attribute   OBJECT IDENTIFIER ::=
      {   generic-sesame-oids   misc-attribute(3) }
     -- OID below which miscellaneous attributes are defined

   primary-group-privilege ::=  OBJECT IDENTIFIER{privilege-attribute 3}


   privilege-attribute OBJECT IDENTIFIER ::=
      {   generic-sesame-oids     privilege-attribute(4) }
     -- OID below which privilege attributes are defined

   qualifier-attribute  OBJECT IDENTIFIER ::=
      {   generic-sesame-oids   qualifier-attribute (4) }
     -- OID below which qualifier attributes are defined

   role-privilege ::=  OBJECT IDENTIFIER { privilege-attribute 1 }

   sesame-key-estb-alg AlgorithmIdentifier ::=  {kd-schemes, NULL }
      -- indicates a SESAME key establishment structure within
      -- an SPKM_REQ structure

   target-name-qualifier OBJECT IDENTIFIER ::=  { qualifier-attribute 1 }

   trust-group-qualifier OBJECT IDENTIFIER ::=  { qualifier-attribute 2 }

   -- Types in alphabetical order

   AccessPrivilegeValueSyntax ::=  Identifier

   AuditIdValueSyntax ::=  Identifier

   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,
   }

   CertandECV ::=    SEQUENCE {
     certificate      [0]   GeneralisedCertificate,
     ecv              [1]   ECV,             OPTIONAL}
            -- ECV is defined in later

   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
        -- only signature supported here
   }

   CommonContents ::=  SEQUENCE{

     comConSyntaxVersion  [0]   INTEGER { version1 (1) }DEFAULT 1,
     issuerDomain         [1]   Identifier             OPTIONAL,
     issuerIdentity       [2]   Identifier,
     serialNumber         [3]   INTEGER,
     creationTime         [4]   UTCTime                OPTIONAL,
     validity             [5]   Validity,
     algId                [6]   AlgorithmIdentifier,
     hashAlgId            [7]   AlgorithmIdentifier    OPTIONAL
   }

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

   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,
     cValues            [1]   CHOICE {
                                encryptedCvalueList  [0] BIT STRING,
                                individualCvalues        [1] CValues
                              }
   }

   ErrorArgument ::=  ENUMERATED {
     gss_ses_s_sg_server_sec_assoc_open                 (1),
     gss_ses_s_sg_incomp_cert_syntax                    (2),
     gss_ses_s_sg_bad_cert_attributes                   (3),
     gss_ses_s_sg_inval_time_for_attrib                 (4),
     gss_ses_s_sg_pac_restrictions_prob                 (5),
     gss_ses_s_sg_issuer_problem                        (6),
     gss_ses_s_sg_cert_time_too_early                   (7),
     gss_ses_s_sg_cert_time_expired                     (8),
     gss_ses_s_sg_invalid_cert_prot                     (9),
     gss_ses_s_sg_revoked_cert                         (10),
     gss_ses_s_sg_key_constr_not_supp                  (11),
     gss_ses_s_sg_init_kd_server_ unknown              (12),
     gss_ses_s_sg_init_unknown                         (13),
     gss_ses_s_sg_alg_problem_in_dialogue_key_block    (14),
     gss_ses_s_sg_no_basic_key_for_dialogue_key_block  (15),
     gss_ses_s_sg_key_distrib_prob                     (16),
     gss_ses_s_sg_invalid_user_cert_in_key_block       (17),
     gss_ses_s_sg_unspecified                          (18),
     gss_ses_s_g_unavail_qop                           (19),
     gss_ses_s_sg_invalid_token_format                 (20)
   }

   ErrorToken ::=    {
      tokenType     [0]   OCTET STRING VALUE X'0300',
      etContents    [1]   ErrorArgument,
   }

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

   GroupPrivilegeValueSyntax ::=  SEQUENCE OF Identifier

   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,
```

```
     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,
     targetAddress        [9]   HostAddress   OPTIONAL
        -- imported from [Kerberos] and used as channel bindings
   }

   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
                             }
   }

   InitialContextToken ::=   SEQUENCE{
     ictContents    [0]   ICTContents,
     ictSeal        [1]   Seal
   }

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

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

   Method ::=  SEQUENCE{
     methodId         [0] MethodId,
```

```
  methodParams     [1] SEQUENCE OF Mparm              OPTIONAL
}
```

```
   MethodGroup ::=  SEQUENCE OF Method

   MethodId ::=  CHOICE{
     predefinedMethod [0] ENUMERATED {
                               controlProtectionValues      (1),
                               ppQualification              (2),
                               targetQualification          (3),
                               delegateTargetQualification  (4)
                             }
   }

      MICToken  ::=    PMToken

   Mparm ::=  CHOICE{
     pValue                 [0]   PValue,
     securityAttribute      [1]   SecurityAttribute
   }

   PACSpecificContents ::=  SEQUENCE{
     pacSyntaxVersion   [0]   INTEGER{  version1 (1)}        DEFAULT 1,
     protectionMethods  [2]   SEQUENCE OF MethodGroup      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
   }

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

   PMTContents ::=  SEQUENCE {
     tokenId              [0]   INTEGER, -- shall contain X'0101' for a MIC
                                         -- token and X'0201' for a Wrap
                                         -- token.

     SAId                 [1]   OCTET STRING,
     seq-number           [2]   INTEGER                 OPTIONAL,
     userData             [3]   CHOICE {
                                plaintextBIT STRING,
                                ciphertext    OCTET STRING
                    }                           OPTIONAL,
```

```
    directionIndicator   [4]   BOOLEAN                OPTIONAL
  }
```

```
   PMToken ::=   SEQUENCE{
     pmtContents    [0]   PMTContents,
     pmtSeal        [1]   Seal
        -- seal over the pmtContents being protected
   }

   PrimaryGroupValueSyntax ::=  Identifier

   PrivilegeAttribute ::=  SecurityAttribute

   PublicKeyBlock ::=  SEQUENCE{
     signedPKBPart  [0]   SignedPKBPart,
     signature      [1]   Signature OPTIONAL,
     certificate    [2]   Certificate OPTIONAL
   }

   PublicTicket ::=  SEQUENCE{
     krb5Ticket     [0]   Ticket,
     publicKeyBlock [1]   PublicKeyBlock}

   PValue ::=  SEQUENCE{
     pv                     [0]   BIT STRING,
     algorithmIdentifier    [1]   AlgorithmIdentifier           OPTIONAL
   }

   Restriction ::=  SEQUENCE {
     howDefined [0] CHOICE {
                hashedExternal  [0] BIT STRING, -- the hash value
                signedExternal  [1] BIT STRING, -- the public key
                certExternal    [2] CertificateId, -- user certificate
                included        [3] BIT STRING
                },
                                -- the actual restriction in a form
                                -- undefined here
     algId      [1] AlgorithmIdentifier              OPTIONAL,
                          -- either identifies the hash algorithm
                          -- or the public key algorithm
                          -- for choices 1 or 2 above.
     type       [2] ENUMERATED  {
                       mandatory   (1),
                       optional    (2)}       DEFAULT mandatory,
     targets      [3] SEQUENCE OF SecurityAttribute    OPTIONAL
   }
                  -- applies to all targets if this is omitted

   RolePrivilegeValueSyntax ::=  Identifier

   Seal ::=  SEQUENCE{
     sealValue          [0]   BIT STRING,
```

```
     symmetricAlgId     [1]   AlgorithmIdentifier      OPTIONAL,
     hashAlgId          [2]   AlgorithmIdentifier      OPTIONAL,
     targetName         [3]   SecurityAttribute        OPTIONAL,
```

```
    keyId              [4]   INTEGER                  OPTIONAL
  }

  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,
    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
  }

  SESAME-AUTHORISATION-DATA ::=  SEQUENCE {
    sesame-ad-type     [0] ENUMERATED  {
                           ppidType  (0)
                         },
    sesame-ad-value    [1] CHOICE  {
                           ppidValue [0]SecurityAttribute
                         }
  }

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

  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
  }
```

```
   --CertificationPath    is imported from [ISO/IEC 9594-8]
```

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

   SpecificContents ::=  CHOICE{
     pac                  [1]   PACSpecificContents
      -- only the PAC is used here
   }

   TargetAEFPart ::=  SEQUENCE {
     pacAndCVs          [0]   SEQUENCE OF CertandECV OPTIONAL,
     targetKeyBlock     [1]   TargetKeyBlock,
     dialogueKeyBlock   [2]   DialogueKeyBlock,
     targetIdentity     [3]   SecurityAttribute,
     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,
  }
```

```
   TrustGroupValueSyntax ::=  Identifier

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

   Validity ::= SEQUENCE {
            notBefore     UTCTime,
            notAfter      UTCTime
   } -- as in [ISO/IEC 9594-8]
   -- Note: Validity is not tagged, for compatibility with the
   -- Directory Standard.

   WrapToken  ::=    PMToken

   END
```

A.2. Kerberos ASN.1 Definitions

The SESAME GSS-API mechanism re-uses the HostAddress and Ticket types from [Kerberos]. These are reproduced here for ease of reference.

```
   SESAME-Kerberos-Definitions   {tbs }

   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}

   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 SESAME mechanism
     forwardable      (1),  -- not supported in the SESAME mechanism
     forwarded        (2),  -- not supported in the SESAME mechanism
     proxiable        (3),  -- not supported in the SESAME mechanism
     proxy            (4),  -- not supported in the SESAME mechanism
     may-postdate     (5),  -- not supported in the SESAME mechanism
     postdated        (6),
     invalid          (7),  -- not supported in the SESAME mechanism
     renewable        (8),  -- not supported in the SESAME mechanism
     initial          (9),  -- not supported in the SESAME 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 SESAME
        -- mechanism.

   END
```

A.3. SPKM ASN.1 Definitions

The SESAME GSS-API mechanism re-uses the SPKM-REQ type from [SPKM]. These are reproduced here for ease of reference.

```
   SESAME-SPKM-Definitions   {tbs }

   DEFINITIONS ::=

   BEGIN

   -- exports everything

   IMPORTS

     AuthorizationData
      FROM SESAME-Kerberos-Defintions   { tbs }

     AlgorithmIdentifier, Certificate, CertificateList,
   CertificatePair, CertificatePath
      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
```

APPENDIX B: Profiling of KD-schemes

The following tables provide profiling information for the data elements defined above and in appendices A.1 and A.2. 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.

B.1. Profile of Ticket as used in symmIntradomain scheme

Field	Value/Constraint
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
-- flags	only bits 6, 10 and 11 can be meaningful in the context of the SESAME 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

B.2. Profile of PublicTicket as used in hybridInterdomain scheme

Field	Value/Constraint
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 SESAME 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
- signedPKBPart	publicKeyBlock
-- encryptedKey	KeyEstablishmentData structure
-- encryptionMethod	sesame-key-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

B.3. Profile of SPKM_REQ as used in asymmetric scheme

Field	Value/Constraint
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 target AEF
- src_name	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 sesame--key-estb-alg
- key_estb_req	contains KeyEstablishmentData with targetApplication field missing
- key_src_bind	missing
req_integrity	sig_integ mandatory
certif_data	only userCertificate field supported
auth_data	missing

Table 4 - SPKM_REQ fields supported

APPENDIX C: ECMA BACKGROUND MATERIAL.

ECMA's work was based on the OSI Architecture [ISO 7498-2], and the series of Security Frameworks developed in ISO/IEC JTC1 [ISO 10181]. A Technical Report, [ECMA TR/46] published in 1988, concentrates on the application layer and describes a security framework in terms of application functions necessary to build secure open systems. The continuation of this report, [ECMA-138], defines the abstract security services for use in a distributed system. A parallel standard, [ECMA-206], describes a model for establishing secure relationships between applications in a distributed system. ECMA has recently completed work to define the functionality and the protocols for a distributed security service in charge of authenticating and distributing access rights to human and application principals, along with supportive key distribution functions. The ECMA standard which is the result of that work is called ECMA-219 [ECMA-219]. It was approved by the ECMA General Assembly in December 1994 and released in January 1995.