Standard ECMA-219
2nd edition - March 1996

ECMA

Standardizing Information and Communication Systems

Authentication and Privilege Attribute Security Application with related key distribution functions

Standard ECMA-219
2nd edition - March 1996

ECMA

Standardizing Information and Communication Systems

Authentication and Privilege Attribute Security Application with related key distribution functions

Part 1: Overview and Functional Model

Part 2: Security Information Objects

Part 3: Service Definitions

Brief History

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

In July 1988, ECMA TR/46, "Security in Open Systems - A Security Framework", was published. In December 1989, based on the concepts of this framework, Standard ECMA-138, "Security in Open Systems - Data Elements and Service Definitions", was produced. It defines a set of Security Services for use in the Application Layer of the ISO OSI Reference Model. Some of these services have been expanded and are incorporated in the present work, though the definitions of the data elements in ECMA-138 have been superseded by this document, and with its publication, ECMA-138 is now withdrawn.

This Standard ECMA-219 describes a model for distributed authentication and access control in which a trusted third party (the "Authentication and Privilege Attribute Application with Related Key Distribution Functions" of this Standard) is used to authenticate human and software entities, provide them with the privileges they need for access control purposes and provide the means of protection of these privileges in interchange. The Standard covers bilateral peer-to-peer authentication, but only in conjunction with key distribution functions.

It defines the security information objects and services that allow Open Systems to interchange authentication information and access control information. The services defined in this document are independent of any particular security policy.

This ECMA Standard is divided in three parts

- Part 1 provides an overview of the Standard, its purpose and content, the functional model that underlies the Standard and its relationships to other standards. In addition, it provides all references for all parts of the Standard.
- Part 2 defines the major Security Information Objects used in the APA-Application.
- Part 3 provides the Service Definitions of the APA-Application. It contains the abstract model, common arguments, common operations and definitions of the Authentication Port and the Privilege Attribute Port.

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

This ECMA Standard has been adopted by the ECMA General Assembly in March 1995.

Table of contents

Part 1 - Overview and functional model

1 Introduction

1.1 Scope

This Standard ECMA-219 defines three applications:

- an Authentication Application,
- a Privilege Attributes Application,
- a Key Distribution Application.

These are distributed Security Applications that provide services concerned with authentication and access control, along with related key distribution information. This definition includes:

- data elements for authentication and access control purposes,
- services for the authentication of human users of computer based systems as well as other active entities,
- services for the provision of Privilege Attributes for purposes of access control in distributed open systems,
- the means of protecting authentication information and Privilege Attributes in interchange between open systems,
- key distribution information for establishing the keys used in this protection,

The last two of these facilities requires that the above applications support a Key Distribution Service, which can either be provided integrally, or as a separate application.

In the interest of providing interoperability between different security systems, the service interfaces are defined such that security information objects not specified in this ECMA Standard but specified elsewhere can be incorporated. One example of such a security system is Kerberos from MIT.

Specification of security policies supporting real-world requirements is outside the scope of this ECMA Standard. It supports, however, a wide variety of such policies. For example identity based, capability based and label based access control schemes.

It is not within the scope of this ECMA Standard to specify subsets (profiles) for implementation and interoperability purposes.

There may be several APA-Applications within a security domain representing different authorities. The interchange of authentication information and Privilege Attributes across security domain boundaries may require an Inter-domain Application, which is for further study.

1.2 Field of application

The field of application of this ECMA Standard is the design and implementation of distributed open systems that support authentication of human users and software entities, and support secure access of users to applications and secure access between distributed applications. Authenticatable users and applications are known as principals.

This ECMA Standard also defines operations which permit clients to obtain cryptographic Keying Information which can be used to secure communications and provide bilateral peer-to-peer authentication.

This ECMA Standard does not define how security data elements are embedded into access protocols defined by other (application) standards (e.g. to support setting up secure associations between clients and servers). However, guidance is provided on how standards developers should make use of the data elements defined in this ECMA Standard.

From an Open Systems viewpoint, the instances of the APA-Applications are Application Entities that are accessed by other Application Entities, the APA-Clients. The APA-Application protocol is implemented by a number of Application Service Elements (ASEs). These ASEs may be combined with other ASEs for other applications (e.g. ACSE, ROSE, Directory, Security Exchange ASE) as required by a given application context. Coordination between the actions of the APA-ASEs and other ASEs is provided by the Single and Multiple Association Control Facilities defined by the OSI Architecture.

The Standard assumes a connection oriented way of working, i.e. the APA-Application is assumed to maintain communications state information. No such requirement is made for security state information however; both "stateful" and stateless ways of working are supported.

From the viewpoint of the general model of "Security Association Management and Support (SAMS)" as defined in [ISO SAMS], the services provided by the A-, PA- and KD-Applications are all realisations of Security Support Services. The ISO Security Support Service Abstract Interfaces are provided by the service interfaces defined in part 3.

1.3 Requirements to be satisfied

1.3.1 Background

This Standard is based on two general requirements that have been considered important: flexibility and ease of management. Flexibility is required in order to support a wide variety of security policies and to allow extensibility. Good security is possible only if it is easily managed, and monitored.

Different authentication methods must be supported. The need for management and control of the logon process, including keeping state information relating to the currently active users of the system, has led to the definition of a stateful on-line Authentication Application.

Access Control Information (ACI) associated with users, in the form of Privilege Attributes, are particularly a target for standardisation since they will be used by many different systems. The standardisation of Privilege Attributes and the transfer of these between systems in a standardised form have received particular attention here. In order to follow the general pattern of OSI standardisation all such specifications are made in ASN.1.

To support as wide a variety of access control methods as possible, as may be required by different security policies it is necessary to provide and manage ACI dynamically and in a timely manner. This requires that the Standard specifies an on-line Privilege Attribute Application. It is thereby also possible to support:

- tempered access rights, i.e. the Privilege Attributes granted to a user being tempered by the properties of the device from which a user is logging on,
- least privilege, i.e. only the minimum rights needed to grant access need be provided,
- anonymous access i.e. a principal's access privileges do not have to reveal the principal's individual identity.

Security information must be protected during transfer. This is done by means of cryptographic techniques, supported by a Key Distribution Service which may or may not be implemented as an application separate from the A- and PA-Applications. In order to control the use of, and prevent the theft of the security information they provide, it is necessary to form a loose coupling between the Authentication Application, Privilege Attribute Application and the Key Distribution Service. As a result this Standard defines all three.

In a distributed environment, proxy (see 4.1.1) needs to be supported. To do this in a controlled manner requires sophisticated protection methods. This Standard has covered the most obvious protection methods, which make sure that Privilege Attributes can only be used as intended. Provision has been made for the addition of other protection methods as they are developed.

See [ECMA TR/46 and ISO 7498/2] for additional background information.

1.3.2 Specific Requirements

The APA-Application is required to:

- authenticate correctly the principals whose verification AI it holds, and validate subsequent incoming requests for service,
- preserve the integrity and where required the confidentiality of that verification AI, and of other related security information,
- provide proper cryptographic protection of its interactions with its clients according to policy,
- produce and integrity protect Authentication and Privilege Attribute Certificates for its clients with contents according to policy,
- respond to management actions according to policy,

- inform Subject Sponsors [ECMA TR/46] of the time-outs in force for the subjects they are sponsoring, according to policy,
- provide cryptographic keys, key certificates and other key objects according to protocol.

1.4 Conformance

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

Type 1	Minimum AS Data Conformance: The implementation shall provide APA-Clients with Authentication Certificates (AUCs) structured and protected according to a subset of the specifications given in clauses 3, 4, 5, 7, 8 and 9 of part 2 of this Standard. This subset must support all of the non-optional fields in the AUC. There is no requirement at this level to support the Keying Information data structures of part 2; Type 1 implementations may provide their own key distribution facilities. The rules in Annex A of part 2, as they apply to AUCs, must be obeyed.
Type 2	Full AS Data Conformance: The implementation shall provide Type 1 conformance, and also provide a subset of the Keying Information data structures described in clause 10 of part 2 of this Standard for use in key distribution to protect the issue of AUCs to PA-Servers.
Type 3	Minimum PAS Data Conformance: The implementation shall provide APA-Clients with Privilege Attribute Certificates (PACs) structured and protected according to a subset of the specifications given in clauses 3, 4, 6, 7, 8 and 9 of part 2 of this Standard. This subset must support all of the non-optional fields in the PAC. There is no requirement at this level to support the Keying Information data structures of part 2; Type 3 implementations may provide their own key distribution facilities. The rules in Annex A of part 2, as they apply to PACs, must be obeyed.
Type 4	Full PAS Data Conformance: The implementation shall provide Type 3 conformance, and also provide a subset of the Keying Information data structures described in clause 10 of part 2 of this Standard for use in key distribution to protect the issue of PACs to productive X-Servers.
Type 5	AS Functional Conformance The implementation is Type 2 conformant, and supports a subset of the operations described in part 3 of this Standard for the Authentication Port, namely those in clauses 4, 5, and 8.1 (as well as the appropriate Bind and Unbind operations). The operations supported must include at least: Authenticate, and DeclareandGetKI. For each operation supported, all non-optional arguments and returns must be supported.
Type 6	PAS Functional Conformance The implementation is Type 4 conformant, and supports a subset of the operations described in part 3 of this Standard for the Privilege Attribute Port, namely those in clauses 4, 6 and 8.2 (as well as the appropriate Bind and Unbind operations). The operations supported must include at least: DeclareandGetKI and DeclareandGetACT. For each operation supported, all non-optional arguments and returns must be supported.
Type 7	Full Conformance Both Type 5 and Type 6 conformance is supported.

1.5 Overview and document structure

1.5.1 Overview of approach taken

This ECMA Standard is designed to offer implementors the ability to implement the security services described at a number of levels of generality. The levels can be pictured as in the "particularisation tree" below.

At the highest level of generality is the ISO Generic Security Services Abstract Service Interface (GSS-ASI), whose definition is out of scope of this ECMA Standard. The PA- and KD-Services defined here are together a specific security mechanism which can be fitted under this ASI (though the method of fitting is not defined). The A-Service defined here is a specific authentication security mechanism which has a wider scope than just the provision of services under the GSS-ASI.

At the next level of particularisation the A-Service is defined with generic operations for exchanging Authentication Information (AI), permitting implementors to describe their own specific AI constructs as they wish.

The A-Service also permits implementors to define their own authentication tokens or certificates that can be returned as a result of successful authentication, but it also defines a specific ECMA Authentication Certificate (AUC) structure designed for use with the ECMA PA-Service.

Similarly, the PA-Service permits implementors to define their own access tokens or certificates, but it also provides a specific Privilege Attribute Certificate (PAC) structure.

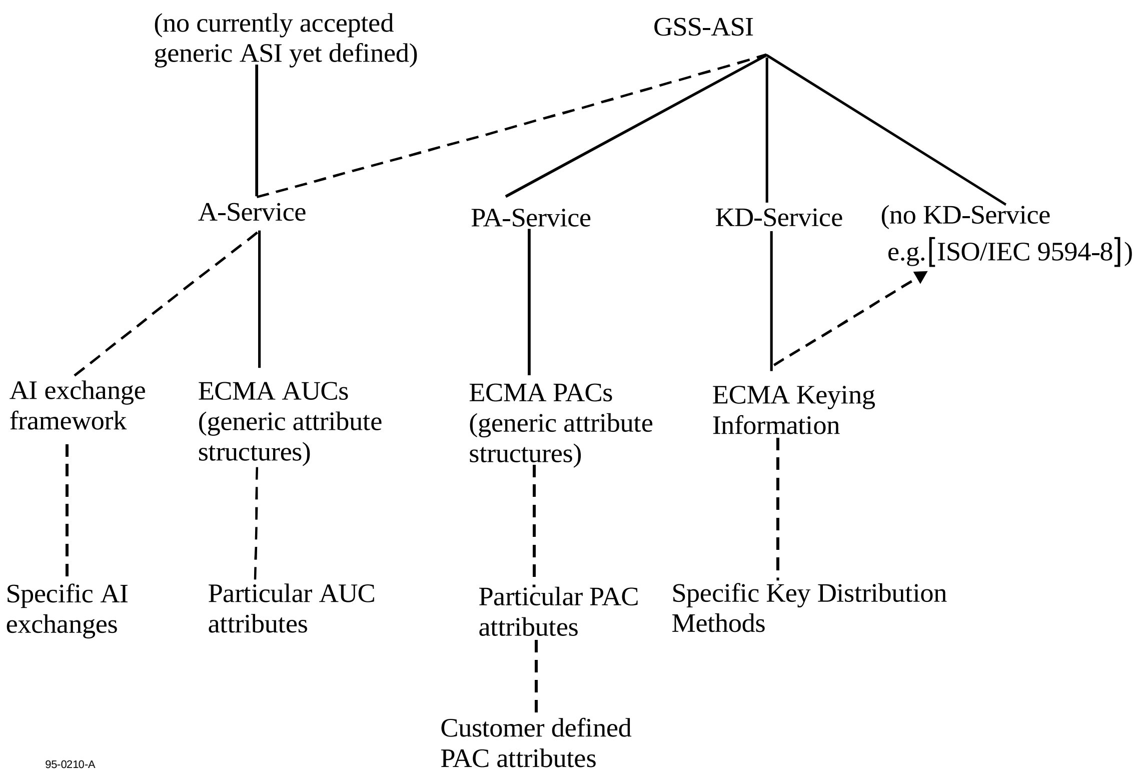
The KD-Service is also defined with a specific Keying Information structure, in which implementors may place specific key distribution constructs to implement a variety of key distribution protocols involving one or more KD-Servers. One of the structures provides the client with the means of doing its own key distribution using public key technology, without further recourse to the KD-Service (for example using the [ISO/IEC 9594-8] Standard).

At the next level of particularisation, specific structures are offered for AI Information, for attributes in AUCs and PACs, and for the contents of Keying Information in support of some particular key distribution methods.

Attribute structures can be implemented (subject to restrictions defined in this Standard) with particular types and contents according to implementor requirements. These types can be registered through the use of OBJECT IDENTIFIERS.

Finally, for PACs it is intended that customers of products implemented to this ECMA Standard should be able to define, register and incorporate their own attribute types in PACs, according to their own security policy requirements.

The ECMA Particularisation Tree


(no currently accepted generic ASI yet defined)
GSS-ASI
A-Service
PA-Service
KD-Service
(no KD-Service e.g.[ISO/IEC 9594-8])
AI exchange framework
ECMA AUCs (generic attribute structures)
ECMA PACs (generic attribute structures)
ECMA Keying Information
Specific AI exchanges
Particular AUC attributes
Particular PAC attributes
Specific Key Distribution Methods
Customer defined PAC attributes
95-0210-A

1.5.2 Document Structure

This ECMA Standard is divided in three parts:

Part 1, (this part) provides an overview of the Standard, its purpose and content, the functional model that underlies the Standard and its relationships to other standards. In addition, it provides references and definitions for all parts of the Standard.

Part 2 contains definitions of Security Information Objects used within this Standard. It gives the syntax and semantics of security attributes, certificates and data structures used to carry cryptographic Keying Information for the establishment of keys for use in protecting communications exchanges . An Annex contains a formal specification of the ASN.1 for the Security Information Objects defined in this part.

Part 3 contains the Service Definitions of the APA-Application. It contains the abstract model, common arguments, common operations and definitions of the Authentication Port, the Privilege Attribute Port and the Key Distribution Port. Annexes to part 3 contain formal ASN.1 specifications, including the use of ASN.1 Object Identifiers, the Object Identifiers themselves, the abstract service specifications and specifications of ASN.1 constructs suitable for use in different authentication methods.

Except for the specification of a management port, the three parts of this Standard together provide a complete description of the external interfaces of the APA-Application. Chaining or referral to other servers of the same APA-Application (see 4.4.3.2) is supported through these interfaces, but the definition of the internal interfaces of such distributed implementations is out of scope of this Standard.

2 References

2.1 Normative references

ECMA-206	Association Context Management including Security Context Management (1993)
ECMA TR/42	Withdrawn - replaced by ISO/IEC 10031-1
ECMA TR/46	Security in Open Systems: A Security Framework (1987)
ISO 7498-2:1989	*Information Processing Systems - Open Systems Interconnection - Reference Model - Part 2: Security Architecture*
ISO 8824:1993	*Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)*
ISO 8825-1:1993	*Information Processing Systems - Open Systems Interconnection - Specification of Encoding Rules for Abstract Syntax Notation One (ASN.1) - Part 1 - Basic Encoding Rules*
ISO/IEC 9072:1989	*Information Processing Systems - Open Systems Interconnection - Remote Operation Syntax, Service Definition and Protocol*
ISO/IEC 9594-8:1990	*Information Processing Systems - Open Systems Interconnection - The Directory, Part 8: Authentication Framework*
ISO/IEC 9595:1991:	*Information Processing Systems - Open Systems Interconnection - Common Management Information Service*
ISO/IEC 9798-2:1994	*Information Technology - Security Techniques - Entity Authentication Mechanisms - Part 2: Entity authentication using symmetric techniques*
ISO/IEC 9798-3:1993	*Information Technology - Security Techniques - Entity Authentication Mechanisms - Part 3: Entity authentication using a public key algorithm*
ISO/IEC 10031-1:1991	*Information Technology - Text and office systems - Distributed-office-applications model - Part 1: General model*

ISO/IEC 10164	*Information Technology - Open Systems Interconnection - Systems Management (a multipart Standard)*
ISO/IEC 10181-1:1992	*Information Technology - Open Systems Interconnection - Security Frameworks in Open Systems - Part 1: Security Frameworks*
ISO/IEC 10181-2:1994	*Information Technology - Open Systems Interconnection - Security Frameworks in Open Systems - Part 2: Authentication Framework*
ISO/IEC 10181-3:	*Information Technology - Open Systems Interconnection - Security Frameworks in Open Systems - Part 3: Access Control*
ISO/IEC 10181-7:	*Information Technology - Open Systems Interconnection - Security Frameworks in Open Systems - Part 7: Security Audit Framework*

2.2 Informative references

Internet RFC 1510	The Kerberos Network Authentication Service (V5) (J.Kohl and C.Neumann, September 1993)
X/Open Preliminary Specification	Generic Security Service API (GSS-API) Base
X/Open Snapshot	Generic Security Service API (GSS-API) Security Attribute and Delegation extensions
ISO SAMS	"Security Association Management and Support (SAMS)", Working Draft 3 from ISO-IEC SC21/WG8 out of the Phoenix USA meeting, December 1994.

3 Definitions and conventions

For the purpose of this Standard the following definitions apply.

The words of defined terms and the names and values of service parameters and protocol fields, unless they are proper names, begin with a lower case letter (e.g. "defined term"). Proper names begin with an upper case letter and are not linked by a hyphen (e.g. Proper Name).

3.1 Imported definitions

The following terms are used with the meaning defined in [ECMA TR/46]:

Control Attribute
Privilege Attribute
security attribute
security domain
Subject Sponsor

The following terms are used with the meaning defined [ISO/IEC 10181.1]:

security certificate
Security Association

The following terms are used with the meaning defined in [ISO/IEC 10181.2]:

Authentication Certificate
claim AI
exchange AI
principal
verification AI

The following terms are used with the meaning defined in [ISO 7498]:

abstract syntax
application-context
application-entity
application-protocol-control-information

application-protocol-data-unit
application-service-element

The following terms are used with the meaning defined in [ISO/IEC 9594]:

chaining
referral
user certificate

3.2 New definitions

See also part 2 Annex A for an expanded description of the different identities defined below.

3.2.1 Access Identity

This is a Privilege Attribute containing an identity used for access control purposes. It is the only type of identity that may be used for this purpose.

3.2.2 Attribute Set Reference

A value recognised by a PA-Server used to identify a group of Privilege Attributes.

3.2.3 Attribute Authority

An authority recognised in a security domain as a trusted source of attributes for entitites within the domain.

3.2.4 Audit Identity

This is an identity attribute containing an identity used only for accountability purposes. Attributes of this type are managed in and issued by the APA-Application. Its purpose is for use in audit trails. The Audit Identity attribute may contain either a name by which the principal which has been authenticated can be directly identified, or a codified value requiring correlation with a recognizable name before such identification is possible. This latter form can be used when the principal requires some form of anonymity.

3.2.5 Authenticated Identity

An identity associated with each principal. There is one Authenticated Identity per principal per Authentication Authority. It is used to identify the claimant for the purpose of authorising the provision of a PAC from an APA-Application.

It is managed by and issued by the APA-Application. It is the identity obtained as a result of successful authentication.

3.2.6 Authentication Level

Authentication Level information is associated with the Authentication Method or combination of methods used by a principal, and in the case of a Secondary Principal, it may be affected by the Primary Principal through which the Secondary Principal's authentication was conducted. It is used to reflect the level of confidence in the results of authentication process.

3.2.7 Authentication Method Identifier

A term used in this Standard to identify authentication based on a single specific mechanism, such as a password exchange. An actual authentication may be performed by means of a combination of methods.

3.2.8 Basic key

A cryptographic key used to provide the protection needed to ensure the secure execution of the system's distributed security operations.

3.2.9 Certificate

A security certificate, as defined in [ISO/IEC 10181-1]

3.2.10 Charging Identifier

The purpose of the Charging Identifier is to correctly charge the user of services and applications. There may be more than one Charging Identifier associated with a principal.

3.2.11 Client Key Block

The part of Keying Information for use by the client that requested it.

3.2.12 Delegate Qualifier Attribute

A security attribute as defined in [ECMA TR/46] which, in a PAC, qualifies an X-Server that can use this PAC only as a delegate. The delegate X-Server cannot use the PAC privilege attributes when authorising access to its own protected resources.

3.2.13 Delegate/Target Qualifier Attribute

A security attribute as defined in [ECMA TR/46] which, in a PAC, qualifies an X-Server that can use this PAC as a delegate or as an accessed target. The delegate/target X-Server can use the PAC by proxy and can use the PAC privilege attributes when authorising access to its own protected resources.

3.2.14 Dialogue key

A cryptographic key used to protect exchanges of application data as required by prevailing security policy. A different dialogue key may be used in the provision of confidentiality from that used in the provision of integrity.

3.2.15 Initiator Qualifier Attribute

An attribute associated with an initiator for the purpose of AUC and PAC control. It may or may not have been guaranteed by a security authority.

3.2.16 Logon Name

This name is used by a principal to describe itself to the system as part of the authentication exchange. A principal may have a number of such names.

NOTE
It is not used as a Privilege Attribute. It is expected that in the case of a human user this name will commonly reflect his real-world name.

3.2.17 Non-repudiation Identity

The Non-repudiation Identity is provided for a human user correctly to understand on whose behalf a digital signature for non-repudiation purposes has been created. An Authenticated Identity for a given entity may have more than one Non-repudiation Identity associated with it.

3.2.18 Primary Principal

A principal that may be used to support the authentication of Secondary Principals. A Primary Principal is a software entity composed in part of an instance of a Subject Sponsor.

3.2.19 Primary Principal Qualifier Attribute

A security attribute as defined in [ECMA TR/46] which, in an AUC or in a PAC, qualifies the security attributes that the initiator's Primary Principal has to possess.

3.2.20 Principal Privileges

Privilege Attributes as defined in [ECMA TR/46], which are used to determine the access rights of their possessor.

3.2.21 Privileges Attribute Certificate (PAC)

A certificate containing Privilege Attributes.

3.2.22 Security Authority

An authority recognised in a security domain as a trusted source of security information.

3.2.23 Secondary Principal

A principal who is sponsored to the system by a Subject Sponsor acting as part of a Primary Principal. It may be a human user or an application entity.

3.2.24 Stateful Server

An APA-Server is said to be stateful for a principal when it maintains security information in the form of a Security Association during the work session of the principal.

3.2.25 Stateless Server

An APA-Server is said to be stateless for a principal when it does not maintain any Security Association between operations.

3.2.26 Target AEF

An Access Enforcement Function, as defined in [ISO/IEC 10181-3] collocated with an X-Server which controls access to that X-Server.

3.2.27 Target Key Block

The part of Keying Information for use by a target AEF with which a cryptographic basic key is to be established.

3.2.28 Target Qualifier Attribute

A security attribute as defined in [ECMA TR/46], which, in an AUC or in a PAC, qualifies an X-Server that can use this AUC or PAC only as an accessed target. The target X-Server cannot use the AUC or PAC by proxy, i.e. it cannot act as a delegate.

3.3 Conventions

This ECMA Standard makes use of the following conventions:

- the abstract syntax definitions use the abstract syntax notation defined in [ISO 8824];
- the distinguished encoding rules defined in [ISO 8825-1];
- the remote operation macros, the application service elements and the application context macros are used as defined in [ISO/IEC 9072];
- the various acronyms for the different services provided by this Standard are used as follows:

APA-Application:	a generic term referring to some or all of the total set of services without presupposing anything about implementation.
*-Server	An implementation of a service on a particular end-system. An A-Server supports authentication operations; a PA-Server supports Privilege Attribute operations and a KD-Server supports key distribution operations. When no particular level of service is being indicated, the term APA-Server is used. X-Server denotes a server with no specified functionality. In particular it may be a normal productive application server.
APA-Client	a client of an APA-Application, with no implied implementation configuration.

3.4 Acronyms

A-	Authentication (Server)
ACF	Access Control Framework
ACI	Access Control Information
ACM	Association Context Management
ADF	Access Control Decision Function
AEF	Access Control Enforcement Function
APA-	Authentication and Privilege Attribute (Application/Server)
AUC	Authentication Certificate
CV	Control Value
DESK	Dynamically Established Shared Key
GSS-ASI	Generic Security Services Application Service Interface
GUC	Generalised User Certificate
KD	Key Distribution (Server)
KI	Keying Information
PA-	Privilege Attribute (Server)
PAC	Privilege Attribute Certificate
PV	Protection Value
SA	Security Association
SAID	Security Association Identifier

4 Functional model

4.1 Environment

In distributed systems, users and the applications they use are distributed over different components of the distributed (or networked) systems. The users in a distributed system may be human beings operating workstations, or application processes running on some open system. Examples of these applications (which are described here as productive applications) are document filing and retrieval, printing, database, and transaction processing. There are many others.

Within such distributed systems there are two generic types of active entity: the client entity and the server entity. Human users use client entities to access information and applications provided by application server entities. Clients and servers are logical entities; a physical system may support multiple client entities and/or multiple server entities. The interaction between clients and servers is mediated by a communications infrastructure. See figure 1.

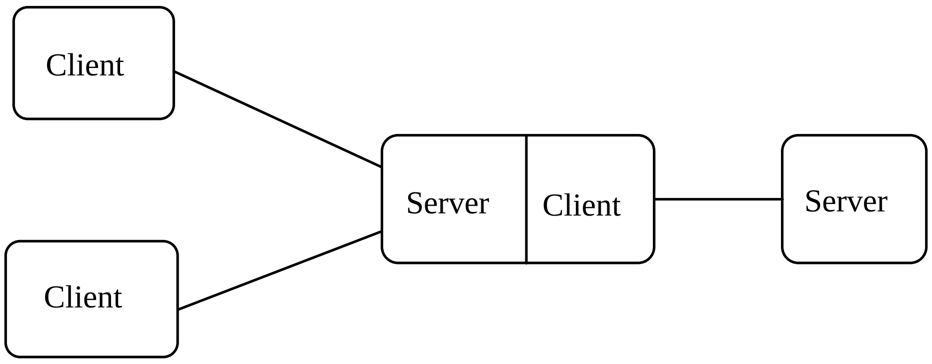


Figure 1 - Client Server Relationships

Productive applications need to protect themselves from unauthorised access and other misuse. In general this protection requires the authentication of their users as well as establishing their access rights relative to the applications they are using. Having each application provide its own protection services that implement authentication and privilege establishment is undesirable because of such considerations as:

- duplication of security information between applications, or conversely the need for users to remember or hold multiple sets of authentication information,
- the need to keep close synchronization of the security information residing on different service providing systems,
- the overhead of managing large numbers of replicated sets of security information,
- the need to secure each of the service providing systems with an adequate level of physical security needed to protect the security information stored on it.

It is more efficient to concentrate security information on one or a few specialised systems and to define a set of security applications for the purpose of the authentication of users, for making their privileges available and enabling them to obtain Keying Information for the protection of communications exchanges. These applications would then act as trusted third parties that mediate between the users and the productive applications. By virtue of this approach, productive applications need not necessarily know about the identity of users and the systems they operate from: they only need to use the privileges supplied by the users in order to make access related decisions. The implication of this approach is that the privileges need to be protected against replay and other misuse. Cryptographic techniques are available to provide such protection.

Authentication operations should be able to be logically separated from Privilege Attribute operations and from key distribution operations so as to allow them to have a different scope and to allow them to operate under different security authorities. It may be convenient and efficient however to implement the server types that support these operations in the same collocated application so they can securely share data to provide improved performance and usability.

4.1.1 Proxy

In distributed systems, in order to provide a service of a particular kind, a server may need to act as a client of servers of other services. For example a user may call a print scheduler to schedule the printing of a file. When the time comes, the scheduler may call a print server to print the file, and the print server may then call a file server to obtain the file to be printed. At each stage appropriate access rights must be established. In this example, the print scheduler may also have access rights of its own with respect to the print servers (the print servers may require that requests are made via the scheduler), but the file server may require the original user's rights to be presented before releasing the file. With respect to the file server the print scheduler and print server are both using the original user's rights by proxy, acting as delegates for that user. At the same time, en route, the print scheduler is adding its own access rights to provide the authority that the print server needs.

Distributed systems also commonly contain application services that are themselves distributed over a number of physical servers. A client accessing such a service may not know precisely which server of the service can support its requests, and may simply make a request on a convenient server, expecting that server to act by proxy for it with respect to another server of the service if necessary. This delegation of access rights could be repeated until the server that can handle the request directly is found.

In some situations use of access rights by proxy is not appropriate and mechanisms must be provided to prevent this.

4.2 Role of the APA-Application

The APA-Application constitutes a trusted third party that provides the above security services. It allows human and application users (principals) to authenticate themselves and to obtain privilege attributes. Principals supply these privileges to the productive applications whenever they need to, for example when they establish a connection and/or with each operation they invoke on the applications. These privileges are contained in certificates (Privilege Attribute Certificates or PACs) that protect the privilege information from modification. Certificate controls and the means to protect exchanges cryptographically are also provided to guard against theft and other misuse (see 4.4.6). Protection of authentication operations and Privilege Attribute operations is ensured by cryptographic keys carried in Keying Information obtained from the KD-Server component of the APA-Application.

The Privilege Attributes provided to the APA-Application's clients convey initiator Access Control Information as described in [ISO/IEC 10181-3], and is used by the Access Control Decision Functions in the distributed system to deny or grant access to applications. See also 5.5.

If authentication operations are supported remotely from Privilege Attribute operations, communication between the two supporting servers is facilitated by the use of an Authentication Certificate (AUC) that can be protected in the same manner as a PAC.

In outline then, the principal is authenticated to an A-Server of the APA-Application, which returns an AUC. The AUC is presented to the PA-Server to authorise the principal's request for a PAC, and finally the PAC is presented to a productive application to authorise the principal's access requests to the application. These steps, and the related key distribution operations, form the main structure around which a number of extensions and variations described in subsequent clauses of this Standard are possible.

4.3 Functional model of the APA-Application

4.3.1 The APA-Application and APA-Clients

Distributed systems provide many productive applications that are used by a wide variety of users. For convenience of discussion these applications are considered to be offered by application server systems and they are accessed by client systems. The client systems are frequently workstations operated by human operators but this is not always the case. Any computer based system may act as a client system in the context considered here. A workstation client can access a server system which acts as a client system to a further server system and so on. Each of these systems require security measures to protect themselves and the data entrusted to them from misuse, contamination and unauthorised observation. These security measures include the authentication of principals (both humans and other), access control and the cryptographic protection of data exchanges. All of these entities are potential users of the APA-Application. However in this document the term "APA-Client" will be used to refer only to the functionality, collocated with the users of the APA-Application, providing the APA-Application service interface.

The components of the functional model are:

- The human user (also known as a Secondary Principal, see below); in many cases the human is the original cause of what is happening.
- The initiator application; any process or function in an active initiator role (also known as a Secondary Principal, see below).
- The Primary Principal; A principal that may be used to support the authentication of Secondary Principals. A Primary Principal is a software entity composed in part of an instance of a Subject Sponsor, and in part an APA-Client. It handles all interaction with the user (human and application).

NOTE
A Primary Principal represents the security environment in which a Secondary Principal is operating; its security attributes are those of that environment. A Secondary Principal possesses security attributes that are independent of the environment in which it exists, but its ability to obtain its attributes may be affected by the attributes of its Primary Principal. A Secondary Principal cannot use an APA-Application without a Primary Principal, though authentication of the Primary Principal is not mandatory.

- The APA-Client; this models the functionality of a given system to act as user of the APA-Application. It is part of a Primary Principal
- The APA-Server; this entity represents an instance of the generic APA-Application defined by this ECMA Standard. In real systems, a multiplicity of these servers may together make up the APA-Application as described in 4.4.4.

The following paragraphs show the relative roles of these components. Figures 2a to 2d show how the APA-Application is described in terms of the Client Server Model.

Figures 2a and 2b show the two views of a distributed application being driven by an application process under the direct control of a human user. Figures 2c and 2d show two similar views, but this time the driving force is a server application.

The terms used to describe the parts of the model to which access is being controlled are given in 4.3.3.

NOTE
The "X-..." terminology in figure 2a is used to indicate generic types and is not to be confused with its use in such contexts as "X-windows".

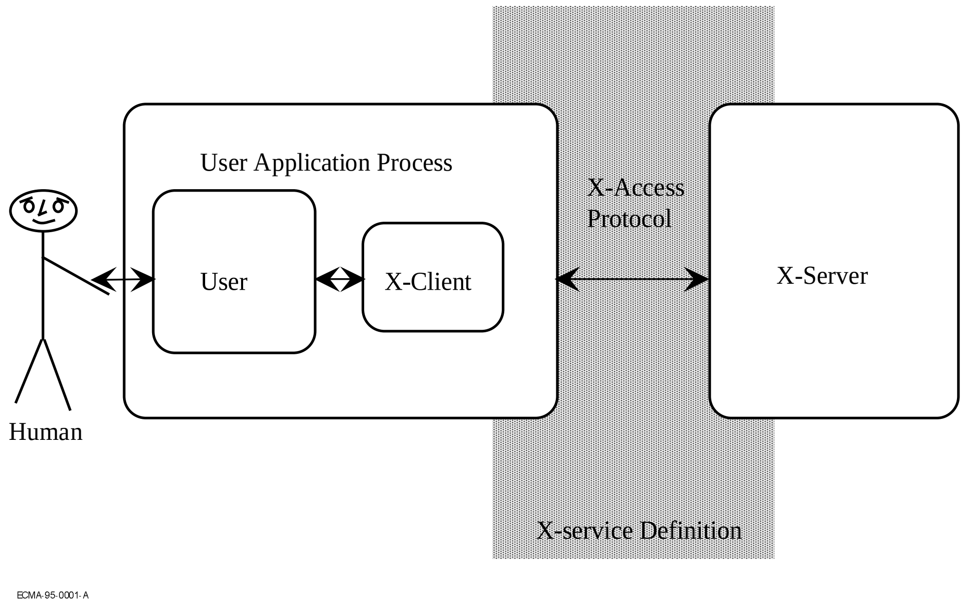


Figure 2a - [ECMA TR/42 - ISO/IEC 10031] View of Human User Driven Distributed X-Application

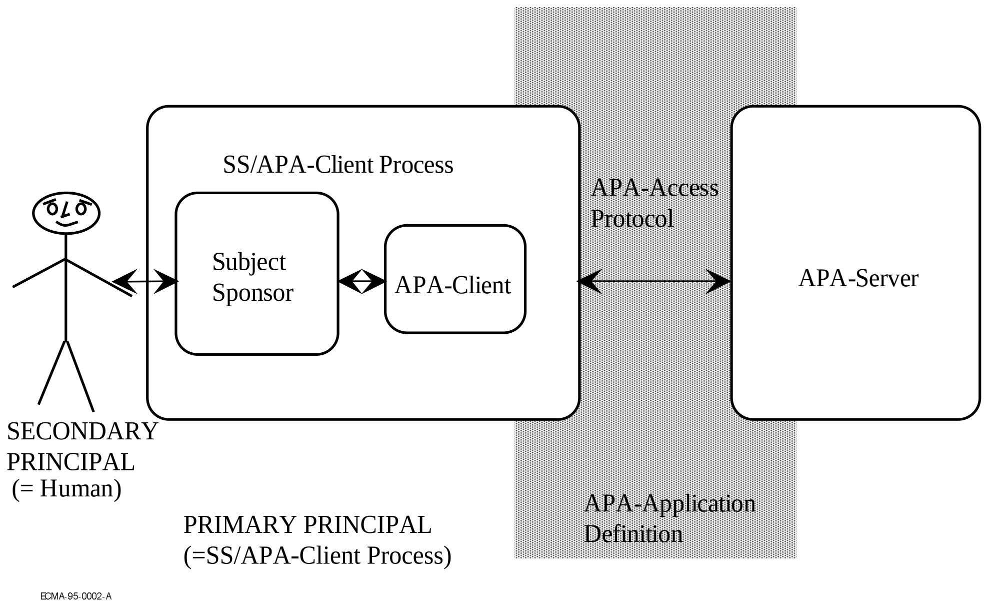

SS/APA-Client Process
Subject Sponsor
APA-Client
APA-Access Protocol
APA-Server
SECONDARY PRINCIPAL (= Human)
PRIMARY PRINCIPAL (=SS/APA-Client Process)
APA-Application Definition
ECMA-95-0002-A

Figure 2b - APA-Application Equivalent to figure 2a

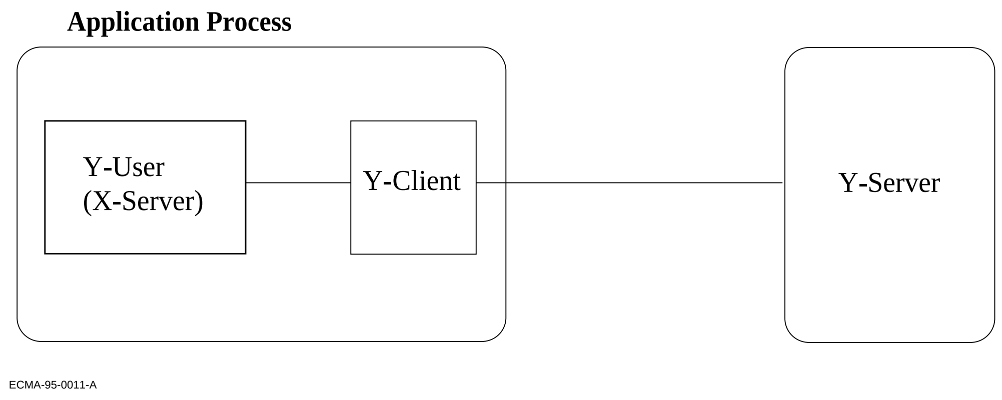


Figure 2c - [ECMA TR/42 - ISO/IEC 10031] View of Application-Driven Distributed Application

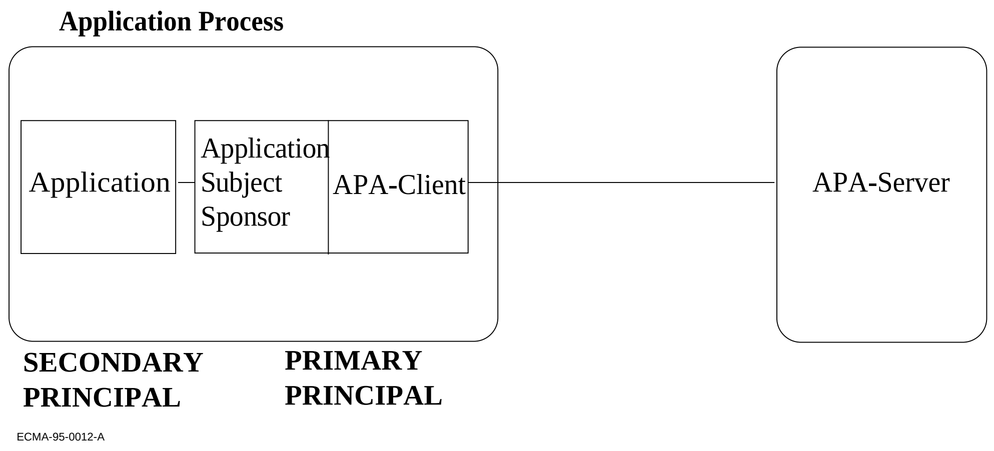


Figure 2d - APA-Application Equivalent to figure 2c.

In principle, access points (as defined in [ISO/IEC 9594]) to an APA-Application may be of different types, providing different combinations of services. The APA-Application provides these different types of behaviour as defined functional subsets. Each access point corresponds to a particular combination of these subsets. These subsets correspond to the ports defined in detail in part 3. Each port defines a particular set of interactions in which the APA-Application can participate with the APA-Client.

4.3.2 APA Abstract Model

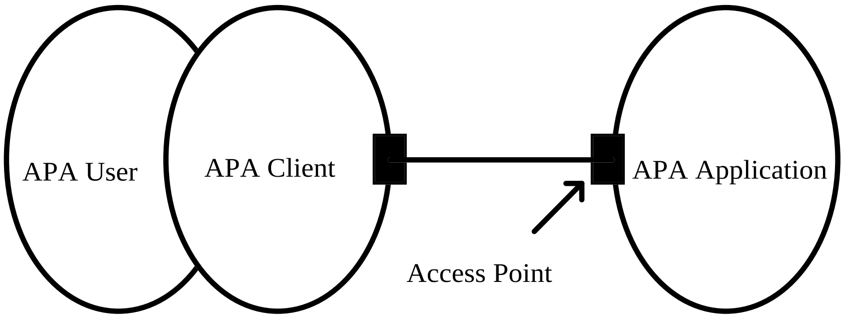


Figure 3. - APA Abstract Model

The abstract model of the APA-Application is concerned only with the external behaviour of the ports. This overall system model is shown in figure 3 above. The complete abstract model is given in part 3 of this Standard. The main components of the abstract model are the APA-Client object and the APA-Application object. Between these objects interaction takes place via ports that are opened at access points on the APA-Application.

4.3.3 Relationship between the APA-Application and other Applications.

The APA-Application is a supportive security application that provides Secondary and Primary Principals with Privilege Attributes and Keying Information. Only if a Principal has authenticated to the APA-Application can it obtain Privilege Attributes from the APA-Application. When a Secondary Principal, e.g. a human user, is using a productive application that is distributed, this is modelled with an X-Client and an X-Server [ECMA TR/42 - ISO/IEC 10031]. This interaction and those between other components of the security infrastructure are illustrated in figure 4.

The human user interacts with the X-Client through the Subject Sponsor. Privilege Attributes may be required for the user to access the X-Client, or any other productive application. Privilege Attributes are also needed by the X-Client for access control purposes when it interacts with an X-Server. These attributes may be its own, those of its user or a combination of both. A simple type of Privilege Attribute is an access identity. In general Privilege Attributes can describe any characteristic of a client or user initiating an access. The different types of privileges required in a specific implementation are defined by the security policy of that particular system. Rules about which initiators obtain Privilege Attributes for access control purposes, what attributes they potentially may require, and the rules for issuing them to the initiators are part of the security policy and not within the scope of this Standard. See also 1.5.1.

In general the X-Server will be concerned with the privileges of the Secondary Principal, the Primary Principal and the X-Client (the initiating entities). Based on the privileges of these entities and the Control Attributes associated with the X-Server it is possible to make an access control decision. The security policy must define what entities and what associated privileges are to be considered when making access control decisions. The security policy also defines how these attributes are managed, how they are associated with principals and the rules that apply when comparing them with Control Attributes for access control decisions. For example, as a minimum solution, it may be sufficient for the X-Client to provide the access identity of the human user Secondary Principal to the X-Server. This identity can then be used if the security policy specifies how an access identity is processed by an Access Decision Function associated with the X-Server to make an access control decision (e.g. using an Access Control List).

This Standard is not concerned with the details of placement and operation of the Access Control Decision Function or the Access Control Enforcement Function. For this see [ISO/IEC 10181-3].

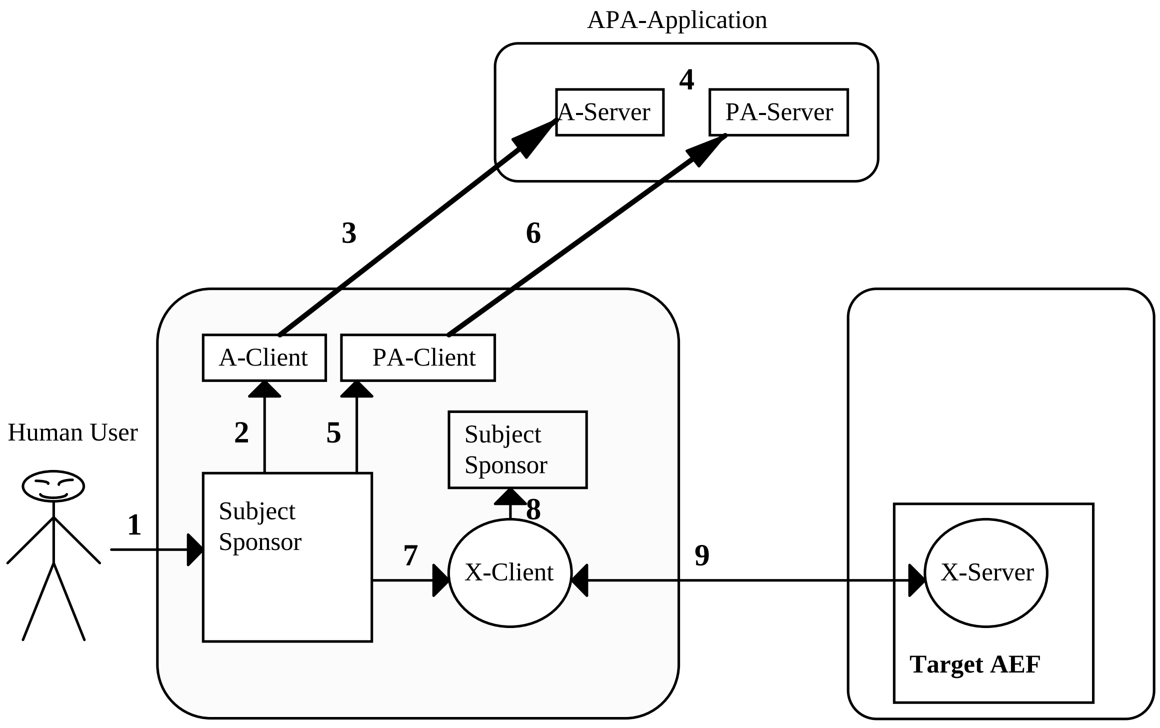


Figure 4 - Relationship with other Applications

An ordering of interactions between different components of the security infrastructure is illustrated in outline in figure 4 with a human Secondary Principal, though others are possible. More detailed walk-throughs are provided in 4.4.8. The totality of possible orderings of APA-Application events between the end-systems shown here is defined in this Standard. However the ordering of internal events is not. Note that the APA-Client object is here split into two different functional units. Key distribution operations are not described here for simplicity. They are described in the walk-throughs in 4.4.8

- The human user interacts with the distributed system through the Subject Sponsor (1).
- The Subject Sponsor initiates authentication of the user through the A-Client (2).
- Authentication of the human user is mediated to the APA-Application and the A-Server authenticates the user (3). The result of the Secondary Principal authentication is returned to the Subject Sponsor via the A-Client.
- After (successful) authentication the A-Server may make available some information directly from the PA-Server when the servers are collocated (4).
- After the A-Client has received the result of successful authentication the Subject Sponsor may be required to obtain Privilege Attributes of the human user if interaction with an X-Client is requested. The Subject Sponsor then requests the privileges of the user from the PA-Client (5).
- The PA-Client will mediate the request for Privilege Attributes to the PA-Server (6). The privileges are issued by an Attribute Authority and returned to the Subject Sponsor via the PA-Client.
- The user's request to interact with the X-Client is handled by an access control decision function (ADF) and an access control enforcement function (AEF) in interaction (7). This is functionality internal to the end-system and can be modelled as part of the Subject Sponsor; the functionality is not within the scope of this Standard. This interaction is a special case of the general model for access control in [ISO/IEC 10181-3]. This general model employs the terms initiator and target for the interacting entities. Access to a target is decided on the basis of initiator Access Control Information (ACI) and target ACI. Privilege Attributes convey initiator ACI and Control Attributes convey target ACI.
- If the X-Client can be authenticated it may have obtained it's own Privilege Attributes in the same way as the human user before any interaction with the X-Server (8).

- The X-Client interacts with the X-Server using the Association Context Management (ACM) service (9), see [ECMA-206]. The outgoing request may also involve an ADF and an AEF. At the target system the incoming request is handled by a target Access Enforcement Function (target AEF). The internal structure of this component, e.g. its use of an Access Decision Function (ADF), and details of its function are local issues.
- The target AEF will need to verify the validity of the Privilege Attributes of the initiator(s) before establishing an interaction. Exceptionally this may require support from the PA-Server through a PA-Client (not shown in the figure).

4.4 Services provided by the APA-Application

4.4.1 Summary of Services

A distinction is made between the logical service types provided by the APA-Application, and the ports through which these services can be obtained. This sub-summarises the operations supported by the APA-Application within each service type. Some of these operations are supported at more than one port. For example the key distribution operations are supported at all APA-Application ports. Details of the mapping of operation to port are given in part 3.

4.4.1.1 Authentication services

The APA-Application supports abstract operations that permit the authentication of Primary and Secondary Principals. The APA-Application acts as an Authentication Authority for the domain in which it operates. The result of successful authentication is an Authentication Certificate (AUC) as defined in this document. This is used to authorise the obtaining of Privilege Attributes.

Authentication related operations are summarised below. They are described in detail in part 3.

Authenticate
This operation is used to authenticate Primary or Secondary Principals, and thereby obtain an AUC.

ContinueAuthentication
This operation is used one or more times to supplement the Authenticate operation if the method of authentication above requires multiple exchanges with the APA-Application.

ChangePassword
This operation enables a principal to change its password through the operational APA-application interface.

ContinueChangePassword
This operation is used to supplement the ChangePassword operation if the APA-Application is required to nominate a list of candidate new passwords from which the principal is to choose one.

CheckAUC
This operation is used by the PA-Application to get an AUC validated.

ConfirmPresence
This operation is used by a stateful A-Application to ask its client whether a principal believed still present there is still actually present. It is used to support policies in which simultaneous access by a principal from two different locations is not permitted. It can also be used in recovery from failure situations.

GetASName
This operation is used when a principal is logging on away from home, and the name of the principal's home A-Server is required.

GetAT
This operation is used to obtain an AUC or other form of Authentication Ticket additional to the AUC returned as a result of successful authentication.

4.4.1.2 Privilege Attribute services

The APA-Application supports abstract operations that provide Privilege Attributes for active entities. These Privilege Attributes are provided in a cryptographically protected Privilege Attribute Certificate (PAC) for use in gaining access to protected resources. The APA-Application acts as an Attribute Authority for the domain in which it operates.

Privilege Attribute related operations are summarised below. They are described in detail in part 3.

CheckPAC
This operation is for use by a target AEF to get a PAC validated.

GetACT
This operation is used to get a PAC or other form of access control ticket

RefinePAC
This operation is used to obtain a PAC with less access privileges and/or more strict controls than an existing PAC. It differs from GetACT in that no authorisation other than the existing PAC (and associated validation information) needs be offered. This operation is intended to be used by targets that can also be initiators acting by proxy, but wishing to use only a restricted form of a received PAC.

4.4.1.3 Key distribution services

The APA-Application supports abstract operations (key operations) that provide Keying Information for establishing cryptographic keys between its clients and other servers. This information can be obtained either as a result of successful authentication to the Authentication Service, or by request to the APA-Application. Keying Information is either used to support the establishment of a Security Association (SA) with an A-, PA- or X-Server or, in the case of connectionless working, is provided by the APA-Client with individual specific operations. The Keying Information enables the establishment of cryptographic keys which are used to provide continuous data origin authentication between the APA- or X-Client and an APA- or X-Server.

Key distribution related operations are summarised below. They are described in detail in part 3.

GetKI
This operation is used to get Keying Information for use in establishing working cryptographic keys with target AEFs.

ProcessKI
This operation is used by a target AEF which receives Keying Information that it cannot process itself, and which must be processed on its behalf by its KD-Server

4.4.1.4 Common services

The APA-Application supports common abstract support operations for the services listed above. These are summarised below.

OpenSA
This operation is used to establish a Security Association between an initiator and a target AEF.

DeclareOperationContext
This operation is used in conjunction with other operations to indicate the security context within which the conjoined operations are being issued. This may either be a reference to an existing SA, or may be fully defined in the DeclareOperationContext operation itself.

CloseSA
This operation is used to terminate an SA. It also indicates that one of the following has happened: the principal concerned has logged off, or is assumed gone away, or is being closed down as a result of management action or is being closed down for security recovery reasons.

RevokeCertificate
This operation is used by the APA-Application to revoke AUCs or PACs that it has issued.

4.4.2 Configuration options

Implementations of the APA-Application may be configured to provide specific combinations of operations to suit operational and security policy needs. This Standard identifies three basic servers: the A-Server, the PA-Server and the KD-Server, all of which can support key operations. Key operation support is provided either by the A- or PA-Server acting as a mediator to a KD-Server, or directly to a separately implemented KD-Server (not shown in figure 5)

4.4.2.1 Key management options

The main data construct used in key distribution is the **Keying Information** construct. Its structure and details of its use are described in 4.5.4. Four uses of Keying Information are supported :

- Keying Information for use in establishing a basic key with an A-Server,

- Keying Information for use in establishing a basic key with a PA-Server,
- Keying Information for use in establishing a basic key with a KD-Server (only when a separate KD-port is used),
- Keying Information for use in establishing a basic key with a target AEF.

Keying Information for use in establishing a basic key with an A-Server, can be :

- either returned by the A-Server itself as a result of a successful "Authenticate" operation,
- or returned as a result of a "GetKI" operation, when a basic key with the KD-Server has already been established (either directly or through a different A-Server or a PA-Server). The name of the A-Server for which key information is to be requested, can be obtained by a "GetASName" operation.

  NOTE
  This happens when Keying Information obtained via the Primary Principal's A-Server is to be used to establish a basic key to protect a Secondary Principal authentication to a different A-Server.

Keying Information for use in establishing a basic key with a PA-Server, can be :

- either returned by an A-Server as a result of a successful "Authenticate" operation,
- or returned as a result of a "GetKI" operation, when a basic key with a KD-Server has already been established (either directly or through an A-Server or different PA-Server). The name of the PA-Server for which key information is to be requested can be obtained by the "Authenticate" operation.

When a separate KD-port is used, Keying Information for use in establishing a basic key with a KD-Server, can be :

- either returned by an A-Server as a result of a successful "Authenticate" operation,
- or returned by a PA-Server as a result of a "GetACT" operation.

The options for the "Authenticate" operation are therefore as follows:

- either it returns a PA-Server name, then Keying Information for use in that PA-Server is later obtained via a GetKI operation,
- or it returns Keying Information for use in a PA-Server (and optionally the PA-Server name).

In either of the above cases, when a separate KD-Port is used, Keying Information for use in the KD-Server can be returned by an "Authenticate" operation or by a "GetACT" operation.

Keying Information for use in establishing a basic key with a target AEF is returned as a result of a "GetKI" operation. The way the target AEF name for which Keying Information is requested is obtained, is outside the scope of this Standard.

NOTE
This Standard does not preclude Keying Information for a KD-Server being obtained exactly in the same way as Keying Information for a target AEF.

4.4.2.2 AUC and PAC options

The A-Server provides authentication services.

The PA-Server provides Privilege Attribute services.

When both of these sets of services are provided for the same principal at the same location, advantage can be taken of the ability for the two servers to share data. Figure 5 outlines these differences.

Separate servers (figure 5A) require that an AUC be used in the first request for a PAC.

Collocation (figure 5B) enables a default PAC to be returned immediately on completion of a successful authentication. When a Security Association is maintained in the APA-Servers (stateful APA-Servers), any subsequent access to the PA-Server for further PACs (the dashed line on the figure shows this to be optional) will no longer require the use of an AUC since the AUC's values can be inferred from the context via a link with the A-Server. The details of these differences are shown in the walkthroughs in 4.4.8.

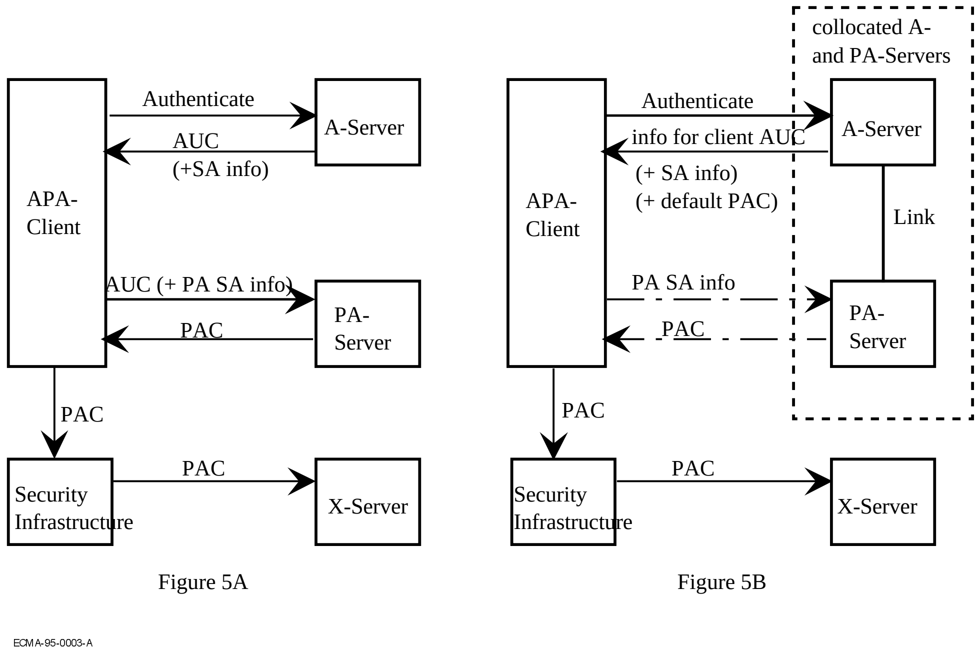


Figure 5 - Relationship Between APA Client and Servers

4.4.3 Distributed working

4.4.3.1 Components of the distributed APA-Application

A distributed APA-Application is defined in terms of a set of one or more APA-Servers which collectively provide distributed authentication and Privilege Attribute applications. This Standard does not provide a specification of the ports that are internal to the APA-Application.

Figure 6 illustrates the distributed components which will be used as the basis for specifying the distributed aspects of the application.

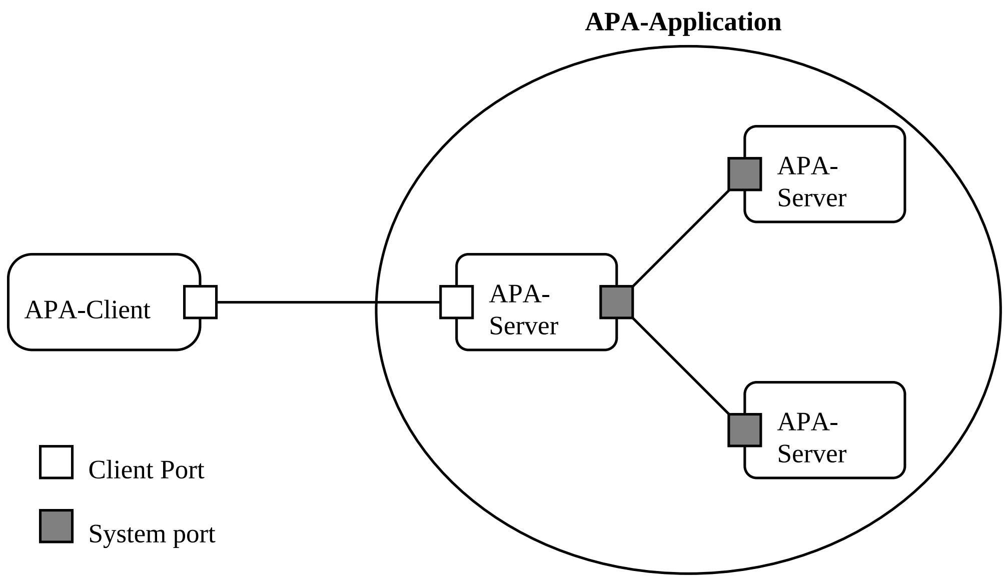


Figure 6 - Objects of the Distributed APA-Application

The APA-Application shares up to three ports with client systems: one primarily for authentication operations, one primarily for Privilege Attribute operations and one dedicated to Key Distribution operations. Further ports (system versions of the client ports) are used for interchange between the distributed components of the APA-Application.

Ports are activated by means of the Bind operation, and de-activated via Unbind. The use of Bind and Unbind in the APA-Application is described in part 3.

4.4.3.2 Chaining and referral

When a Secondary Principal is unknown to the A-Server of the Primary Principal through which the system is accessed (for example a human user logging in away from his normal base), the A-Server may support either of two different types of mutually independent service: chaining or referral.

When referral is supported, the A-Server will return a reference to one or more A-Servers that may be able to handle the request it could not itself handle. The referenced A-Server is then accessed directly by the Primary Principal. Irrespective of whether the "home" APA-Application is implemented as separate or collocated A- and PA- servers, the communications that follow are made directly from the Primary Principal to the A-Server and PA-Server.

When chaining is supported, the A-Server will relay the request to another A-Server. If this latter is able to handle the request, following successful authentication it returns a reference to the PA-Server (as usual) and if the home A- and PA- servers are collocated, possibly a default PAC. Further communication with the PA-Server is performed directly. Communication with the home A-Server with which the Security Association has been established continues to be relayed via the local one.

Servers which chain and Servers which refer can be invoked in the same authenticate operation. For example an A-Server may relay a request to an A-Server which refers. However some APA-Clients may not support referral and are only able to connect to one local A-Server. In this case this local A-Server must handle chaining and referral in such a way that the APA-Client it serves is never referred to another server. This can, for example, be supported by including in the protocol message a flag indicating whether the calling entity (APA-Client or relaying A-Server) supports referral or not. This flag is outside the protected message and may be changed at each leg of the chain.

4.4.4 Modes of operation

When security information on a principal is maintained in an APA-Server, in the form of a Security Association during the work session of the principal, the server is said to be stateful for this principal. Otherwise, it is said to be stateless. This Standard enables servers of the APA-Application to be either stateful or stateless for a given principal. This property is independent of the ability of the APA-Server to maintain state information relative to a specific operation. For example, a stateless A-Server may be able to handle state information related to multiple authentication exchanges (e.g. a challenge in a challenge/response authentication).

4.4.4.1 The use of operation contexts

The operation context contains parameters that enables operations to the APA-Application to be protected and authorised, whether the APA-Servers are stateful or stateless.

Continuous security protection provided by this Standard for messages passed between APA-Servers and their clients is provided on an operation context basis.

For stateful APA-servers the operation context refers to a Security Association (SA), which is that part of an Association Context, as defined in [ECMA-206], dealing with security. SAs can be established with an A-Server after successful authentication. SAs with a PA-Server can be established from information provided from the A-Server. SAs with a KD-Server can be established either from information provided from an A-Server or a PA-Server or by information directly shared between the initiator and the KD-Server.

SAs are not supported in stateless APA-Servers. For such servers the operation context must carry the protection information itself.

Operation contexts are specified, either by reference to an SA or directly, using Declare OperationContext (see part 3).

4.4.4.2 The use of Security Associations

The APA-Application supports the use of an SA to permit a stateful security relationship to be established between a Primary and/or Secondary Principal and a server of an APA-Application. This relationship is independent of any underlying OSI association or connection that might be established between them and the APA-Server. An SA of a Secondary Principal can optionally be linked to one of a Primary Principal.

An SA may have associated with it one or more cryptographic keys (as described in 4.5.4) under which messages exchanged as part of the association are protected. The protection takes the form of integrity seals to provide data origin authentication and integrity, optionally supplemented by encryption for confidentiality. The actual way in which this protection is implemented, in particular the OSI protocol layer in which it is done, is not dictated by this Standard, though if done at a lower layer, there needs to be a one-to-one relationship between the application and the lower layer communicating entities.

An SA may also have associated with it other security information of the kind that can be passed in an AUC or PAC.

The APA-Application supports the establishment of SAs in a number of ways:

- in the case of an SA between a client and an A-Server, a key and Security Association information can be derived from the authentication process. The method of deriving the key is specific to the authentication method in use; the other context information is taken from the AUC returned by the A-Server.
- in the case of an SA between a client and a PA-Server, a key can be established by the client first obtaining Keying Information from the A-Server as described in 4.5.4. Other SA information can either be provided by the presentation of an AUC under the protection of the above key, or in the case of a PA-Server collocated with an A-Server, by being able to link the SA, via Keying Information, with the SA already established with the A-Server as described in 4.4.2.2.
- in the case of an SA between a client and a KD-Server, a key can be established by the client first obtaining Keying Information, either provided from an A- or a PA-Server or by the initiator directly using installed values. No additional authorisation information is used.
- in the case of an SA established in relation to a previously established SA (the situations in which this can happen are described below), a key derived from the first SA's key can be used, or the same key can be specified for use within the new SA.
- the APA-Application can also provide Keying Information with which an SA can be established with the target AEF of a normal productive X-Server application component.

Specific combinations of uses of SAs by Primary and Secondary Principals are described below.

4.4.4.2.1 No Primary Principal Keying Information

The Primary Principal may not be able to or may not choose to establish a key of its own with an APA-Server, for example it may not be registered with the A-Server, PA-Server or KD-Server.

In such a case when a Secondary Principal uses the Primary Principal to access an APA-Server, it can establish a Secondary Principal SA with no Primary Principal Keying Information. It can do this by authenticating itself (in the case of an A-Server) or explicitly via an Open SA operation. The SA has a key associated with it , and messages sealed under this key for data origin authentication and integrity purposes can as a result be associated with the Secondary Principal. The SA established is from the Secondary Principal only, and any claimed security properties of the Primary Principal must be explicitly presented via an AUC or PAC of the Primary Principal.

4.4.4.2.2 Primary Principal Security Associations

Alternatively a Primary Principal may be able to establish an SA by authenticating itself to an A-Server in a manner which establishes a shared key with the A-Server, or by use of Open SA as described above.

If a Primary Principal has established an SA with an APA-Server, it may then establish a number of simultaneous Secondary Principal SAs with the same server (provided that the server recognises the Secondary Principals involved), linked to the SA already established, as described below. Each Secondary Principal SA may have a separate key under which it is protected (either derived as a function of the Primary Principal SA key (possibly in combination with a value obtained as a result of authentication of the Secondary Principal), or obtained from the APA-Application via a Get KI operation as described in part 3), and messages sealed under this key for Data Origin authentication and integrity purposes can be associated with the authenticated Secondary Principal in a way which is linked to its use of the Primary Principal.

4.4.4.2.3 Primary Principal Operation Contexts

In the same way, a Primary Principal may also be able to establish operation context Keying Information even if it authenticates without establishing an SA.

It may then establish a number of simultaneous Secondary Principal SAs with the same server (provided that the server recognises the Secondary Principals involved), based upon this Keying Information

4.4.4.2.4 Security Association control

An SA is identified by means of two SA Identifiers (SAIDs), one owned by the principal concerned and the other owned by the APA-Server. The first is unique to the principal, the second to the APA-Server. Each has a value determined by and convenient for its owner.

An SA with an A-Server is initiated as a result of a successful authentication of the principal involved. A PA-Server SA is initiated by means of an Open SA operation. An SA's existence is independent of any other forms of association that may exist between the two parties. It may have a life which extends over multiple Binds, or which is shorter than the lifetime of a single Bind. An SA is terminated by either:

- a Close SA operation,
- various failures (identified in the detailed operation specifications),

A Close SA operation can be issued by either the APA-Client or the APA-Application, and may be, for example, as a result of a user logging off, expiry of a period of inactivity, or a recovery action.

4.4.4.3 Certificates in conjunction with Security Associations

At any time during the life of an SA a certificate (AUC or PAC) can be presented to support authorisation of operations over the SA and establish Security Association information for succeeding operations. If the operation with which a certificate is presented is an Open SA operation, then succeeding operations in the SA are treated as authorised under the certificate. If presented with any other operation the certificate is simply used in support of that operation and has no longer term effect on the SA. Thus for example when accessing a PA-Server, an AUC could be presented by means of an Open SA operation once at the start of an SA established with the server to authorise all subsequent requests for PACs. Subsequently a different AUC could be presented via a second Open SA operation, changing the context information and consequently the authorisation and associating it with the new SA. Alternatively, an SA could be established between a Primary Principal and a PA-Server. Different Secondary Principals requesting PACs from the PA-Server could use that SA but always present their AUC in each PAC request.

4.4.4.4 Certificates without explicit communication cryptographic protection

In this mode the SA is not used. Instead, only an AUC is obtained by the client as a result of successful authentication of the principal. Either this AUC or a subsequently obtained PAC is subsequently passed with each APA- operation in order to authorise it according to what the operation requires (see part 3).

This mode of operation can be used when the communications channel between the APA-Client and APA-Application is secured with origin authentication and integrity protection by other means than those described in this Standard.

NOTE
If no communication protection is provided in this mode of operation, communication is susceptible to wire-tap attacks, and information from the APA-Application can be compromised.

4.4.5 Use of cryptographic features in clients

The security applications defined by this Standard accommodate APA-Client systems with cryptographic capabilities employing asymmetric algorithms, symmetric algorithms or both, as well as those without cryptographic capabilities. It supports clients hosted in computers both with and without key stores.

Where cryptographic facilities are available in client systems, cryptographic protection can be obtained for exchanges between the APA-Client and APA-Servers or other servers.

When the client system also contains a secure key store it may be able to authenticate itself as a Primary Principal to the APA-Application.

Some forms of authentication supported by this Standard require the use of one way functions in clients. They are also needed for certain forms of Keying Information processing (see part 2), and for some of the certificate control methods.

4.4.6 Certificate Controls

A variety of methods are provided to control the use of AUCs and PACs (referred to as "certificates" below). There are methods of preventing the certificate from being stolen from its legitimate users, and from being misused by its legitimate users. The methods are summarised below. They are described in detail in part 2.

Prevention from stealing

- the ways in which a certificate can be used by proxy can be controlled by means of a Protection value in the certificate. This is used in conjunction with an encrypted "Control Value" sent with the PAC (see 4.5.3),
- proxy can also be more tightly controlled, with each X-Server application component in the access chain being able to trace the full proxy path by means of protection fields in the certificates used.
- similar controls can be imposed by identifying in the certificate the valid users of that certificate, either by name or some other attribute. The values found will then be compared at the target AEF with corresponding information obtained via cryptographic key distribution schemes or by other means.

Prevention from misuse

- the certificate can be constrained to be valid only with respect to target X-Server application components possessing certain security attributes,
- the certificate can be constrained to be valid up to a certain expiry time, and then only at certain times within that period,
- a count of the number of times a certificate can be used at a particular target AEF can be specified,
- a certificate can require that any target AEF to which it be offered should check back with the issuer to ensure that it is still valid,

4.4.7 Certificate revocation

There are a number of ways in which certificates issued from the A- or PA-Application can be revoked:

- if an SA is terminated in the normal course of events, for example following a user log off, any communications associations or SAs established on the basis of certificates obtained through the closed SA are expected to be closed, and associated copies of certificates held in the APA-Client System destroyed, but no special recovery action is expected of the APA-Client system,
- if an SA is terminated (usually by the APA-Application) for security recovery purposes, the APA-Client system is expected to perform the actions above, but is further expected to transmit to the targets of any terminated communications associations or SAs, the fact that the termination was abnormal, with the implication that appropriate recovery action should be taken in the target system. The ways in which this may be done with respect to target application components other than APA-Application targets is out of scope of this Standard, but it can be seen that a cascade effect is possible, with each target passing on the bad news to the next. With respect to A- or PA-Server targets it is sufficient to unbind any extant Binds and terminate the SA using Close SA indicating recovery as the reason.
- If a PA-Server issues a Revoke-Certificate operation, this is expected to have the same effect as an SA recovery termination, but confined only to communications associations and SAs with which the certificate(s) in question have been used.

4.4.8 Walk-throughs

Three walk-throughs are given below;

- the first describes the simple case of a Secondary Principal authenticating and obtaining its PAC from separate A- and PA-Servers,
- the second is as above but with the A- and PA-Servers collocated
- the third describes the case where the Secondary Principal is using a Primary Principal which itself has been authenticated to the same or separate A-Server.

The main operations used at each step are given in brackets at the appropriate points. These operations are specified in full in part 3. Not explicitly described in the walkthroughs is the use of the DeclareOperationContext operation, which provides replay protection and either defines the SA within which the operations it is combined with are being initiated or provides Keying Information to protect the operation it is combined with. part 3 describes this, and also shows how the main operations may themselves be combined to reduce the number of communications exchanges. For simplicity the walkthroughs describe each operation as a separate step.

The walkthroughs below describe the option where Keying Information for use in the PA-Server is returned by the A-Server and where the KD-Server is accessed through the A- or the PA-Server (i.e. it is not a separate port).

4.4.8.1 Secondary Principal Authentication - Separate A- and PA-Servers

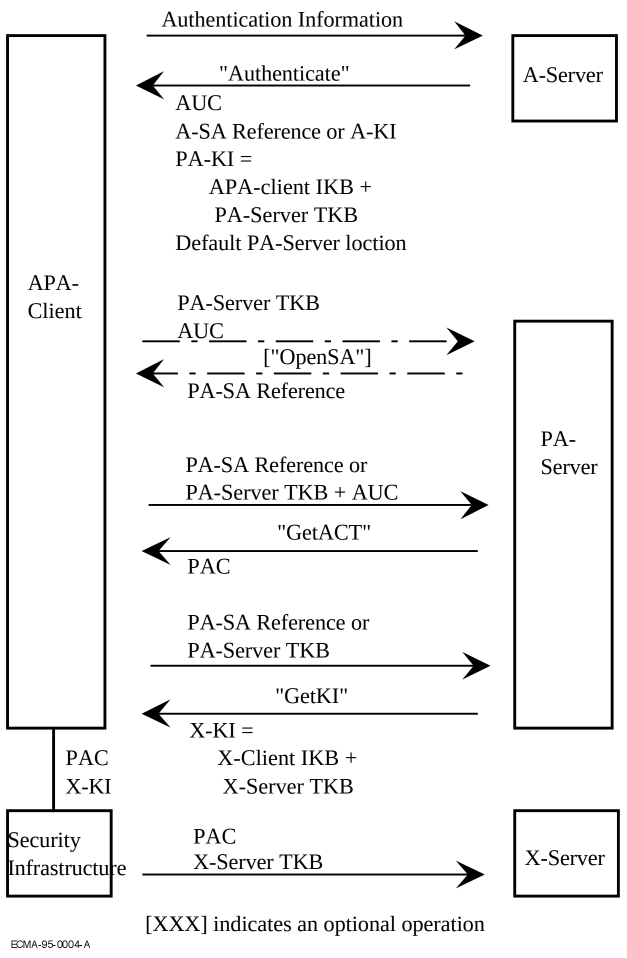


Figure 7 - Secondary Principal authentication - Separate A- and PA-Servers

1. The APA-Client is used to authenticate the Secondary Principal to the A-Server (**Authenticate** and possibly **ContinueAuthentication**)
2. If this is successful the A-Server returns:
   - an AUC, which may be either sealed or signed to protect it from being tampered with in the APA-Client system before accessing a PA-Server. If sealed the AUC is intended for use with a particular PA-Server sharing the seal's key with the A-Server. If signed it can be used with any PA-Server which provides privileges for this APA-Client,

     if the A-Server is stateful, the authentication operation also automatically establishes an SA between the APA-Client and the A-Server in support of future operations on the A-Server (e.g. for logging off), the AUC also provides the caller with the Audit Identity of the Secondary Principal and the time at which the authentication expires. Security policy and other information may also be separately returned (not shown on the figure) such as the inactivity time-out period for this principal, or just user news,

- if the A-Server is stateful, a reference to the above SA (A-SA Reference) for use by the APA-Client,
- if the A-Server is stateless, Keying Information (A-KI) to protect future operations on the A-Server. This information is in the form of two cryptographic Keying Information packages (an Initiator Key Block intended for the APA-Client and a target key block to be sent in any future operation).
- Keying Information (PA-KI) to help in securing interactions between the APA-Client and the default PA-Server for this client. This information is in the form of two cryptographic Keying Information packages, one intended for the APA-Client : the initiator key block (APA-Client IKB), the other for the default PA-Server : the target key block (PA-Server TKB).
- the location of the default PA-Server.

3. If the PA-Server is a stateful server, the APA-Client uses the AUC and Keying Information (PA-KI) to establish an SA with the PA-Server to authorise and protect its exchanges with it (**OpenSA**).

   It then requests a PAC (**GetACT**) and requests Keying Information (**GetKI**) for accessing a specified X-Server application (i.e. one with which the PAC is going to be used).

   When the PA-Server is a stateless server, the AUC is sent in the GetACT request to authorise the operation, and the target key block (PA-Server TKB) for the PA-Server is sent in each operation to the PA-Server to protect the operation.

4. The PA-Server returns:

   - in response to GetACT, the PAC and any control information needed for it,
   - in response to OpenSA, its own reference for the SA just established with itself, for future use by the client when performing further PA-Server operations,
   - in response to GetKI, Keying Information (again in the form of two key blocks) to enable any future interactions between the APA-Client and the identified X-Server's application to be secured.

5. Out of scope of this Standard, the PAC and Keying Information are deposited with the security infrastructure, which then can use them to authorise and secure access to the X-Server application.

4.4.8.2 Secondary Principal Authentication - Collocated A- and PA-Servers

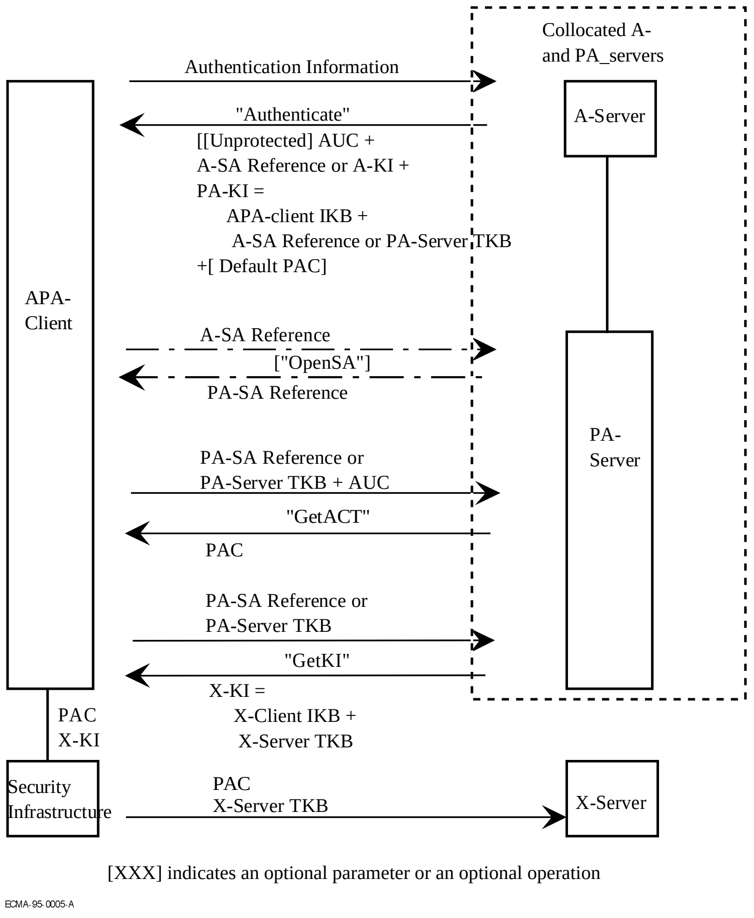


Figure 8 - Secondary Principal authentication - Collocated A- and PA-Servers

1. The APA-Client is used to authenticate the Secondary Principal to the A-Server (**Authenticate** and possibly **Continue Authentication**)
2. If this is successful the A-Server returns:
   - an AUC that, if the A-Server is stateful, does not need to be sealed or signed (though a signed or sealed AUC could also optionally be returned for use with respect to additional PA-Servers which are not collocated with the A-Server),

     if the A-Server is a stateful server, the authentication operation also automatically establishes an SA between the APA-Client and the A-Server in support of future operations on the A-Server (e.g. for logging off), The AUC information which is a part of this context is also held in a data store accessible from the collocated PA-Server,

     the AUC also provides the caller with the Audit Identity of the Secondary Principal and the time at which the authentication expires. Security policy and other information may also be separately returned (not shown on the figure) such as the inactivity time-out period for this principal, or just user news,
   - if the A-Server is a stateful server, a reference to the above SA (A-SA Reference) for use by the APA-Client,
   - if the A-Server is a stateless server, Keying Information (A-KI) to protect future operations on the A-Server.
   - Keying Information (PA-KI) in the form of an initiator key block intended for the APA Client (APA-Client IKB) and a target part intended for the PA-Server.

- If the APA-Servers are stateful, the target part is a reference to the SA between the APA-Client and the A-Server (A-SA Reference). Otherwise, it is a target key block (PA-Server TKB),
- an optional default PAC provided from the PA-Server's database.

NOTE
A stateless A-Server that does not return a default PAC does not take advantage of the collocated APA-Servers.

3. If the default PAC received is acceptable for use with the target X-Server application component about to be accessed, the APA-Client needs only obtain the Keying Information (X-KI) to secure the access to it. The APA-Client may already have done this, or may have obtained the Keying Information separately by means outside the scope of this Standard, in which case, the APA-Client may now skip steps 4 and 5.
4. If the PA-Server is a stateful server, the APA-Client uses the Keying Information (PA-KI) to establish an SA with the PA-Server to authorise and protect its exchanges with it (**OpenSA**). The PA-Server uses the reference found in the received SA information to identify the AUC information to be associated with this context.

   The APA-Client then requests a PAC (**GetACT**) and requests Keying Information X-KI (**GetKI**) for accessing a specified X-Server application (i.e. one with which the PAC is going to be used).
5. As in the previous walk-through, the PA-Server returns:
   - in response to GetACT, the PAC and any control information needed for it,
   - in response to OpenSA, its own reference for the SA just established with itself, for future use by the client when performing further PA-Server operations,
   - in response to GetKI, Keying Information X-KI (again in the form of two key blocks) to enable any future interactions between the APA-Client and the identified X-Servers application to be secured.
6. Out of scope of this Standard, the PAC and Keying Information are deposited with the security infrastructure, which then can use them to authorise and secure access to the X-Server application.

4.4.8.3 Secondary Principal authentication in the context of security properties of a Primary Principal

1. The APA-Client is used to authenticate the Primary Principal to an A-Server, establishing an SA (PP A-SA) with it or getting Keying Information (PP A-KI) to protect future operations (as described for Secondary Principals either in 4.4.8.1 for separate A- and PA- Servers or in 4.4.8.2 for collocated ones).
2.

   2a When the same A-Server authenticates both the Primary Principal and the Secondary Principal, the Primary Principal uses the APA-Client to authenticate the Secondary Principal to the A-Server (**Authenticate** and possibly **ContinueAuthentication**). If an SA has been established during the Primary Principal authentication, this PP A-SA is used to protect the Secondary Principal authentication. Otherwise, the Secondary Principal authentication is protected by a key obtained from PP A-KI.

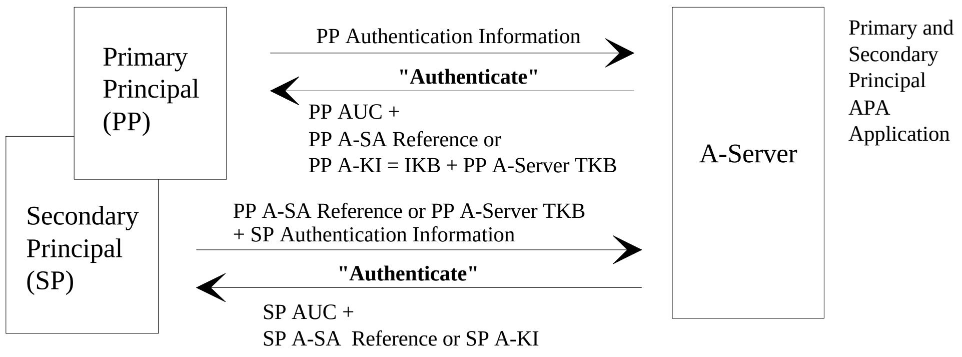

Primary Principal (PP)
Secondary Principal (SP)
PP Authentication Information
"Authenticate"
PP AUC +
PP A-SA Reference or
PP A-KI = IKB + PP A-Server TKB
PP A-SA Reference or PP A-Server TKB
+ SP Authentication Information
"Authenticate"
SP AUC +
SP A-SA Reference or SP A-KI
A-Server
Primary and Secondary Principal APA Application

Figure 9a - Secondary Principal authentication in the context of an authenticated Primary Principal - Same A-Server

NOTE
Subsequent authentication of Secondary Principals using the same Primary Principal to this A-Server can be performed within the same PP A-SA or protected by the same PP A-KI.

2b When the Secondary Principal authenticates to an A-Server different from the one that authenticates the Primary Principal, the Primary Principal has to find the name of the Secondary Principal A-Server to get Keying Information for it :

- it uses the APA-Client to obtain the name of the Secondary Principal A-Server (**GetASName**),
- it obtains Keying Information (AS2-KI) for this A-Server (**GetKI**)
- the APA client uses this Keying Information to protect the Secondary Principal authentication,

  If the Secondary Principal A-Server is stateful, the APA-Client creates an SA for the Primary Principal (PP AS2-SA) to protect the Secondary Principal authentication (**OpenSA**), otherwise, the Primary Principal Keying Information (PP AS2-KI) is directly used in the authenticate operation to protect the Secondary Principal authentication.

  Following a successful authentication (**Authenticate** and possibly **ContinueAuthentication**), an SA for the Secondary Principal may also be created (if the Secondary Principal authentication is stateful).

NOTE
In the following figure, either PP A-SA Reference or PP A-KI has been previously returned by PP's A-Server along with PP PA-KI after the PP authentication. A PP PA-SA may also have been established with the PP's PA Server (see 4.4.8.1 or 4.4.8.2)

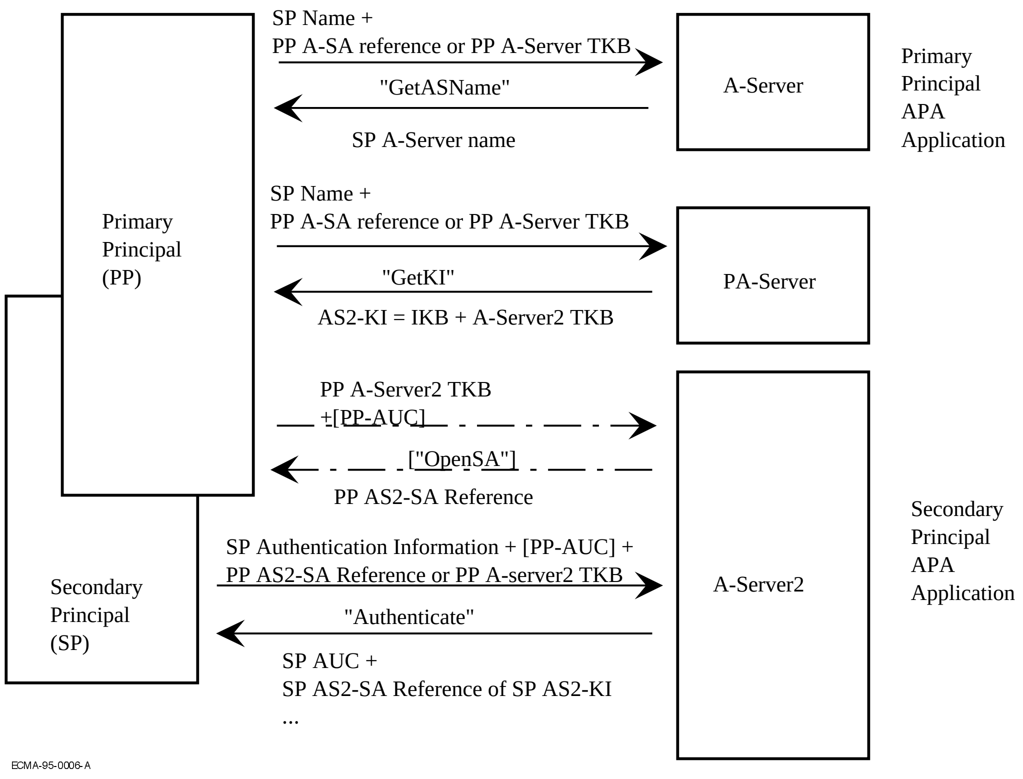

SP Name +
PP A-SA reference or PP A-Server TKB
"GetASName"
SP A-Server name
A-Server
Primary Principal APA Application
SP Name +
PP A-SA reference or PP A-Server TKB
"GetKI"
AS2-KI = IKB + A-Server2 TKB
PA-Server
Primary Principal (PP)
PP A-Server2 TKB
+[PP-AUC]
["OpenSA"]
PP AS2-SA Reference
SP Authentication Information + [PP-AUC] +
PP AS2-SA Reference or PP A-server2 TKB
"Authenticate"
SP AUC +
SP AS2-SA Reference of SP AS2-KI
...
A-Server2
Secondary Principal APA Application
Secondary Principal (SP)
ECMA-95-0006-A

Figure 9b - Secondary Principal authentication in the context of an authenticated Primary Principal - Different A-Servers

NOTES

1) *Subsequent authentication of Secondary Principals to A-Server2 (using the same Primary Principal) can be performed within the same PP AS2-SA or protected by the same PP AS2-KI.*

2) *Alternatively the Primary Principal may decide not to use its own capabilities in support of establishing an SA within which to authenticate the Secondary Principal. The process would then be as described in 4.4.8.1 for separate A- and PA-Servers or in 4.4.8.2 for collocated ones.*

Once the Secondary Principal is authenticated to its A-Server, the sequence of operation is not affected by the Primary Principal's presence (the steps are as in 4.4.8.1 for separate A- and PA- Servers or as in 4.4.8.2 for collocated ones). The Primary Principal's AUC or PAC respectively may however have beneficially affected the Secondary Principal's AUC and PAC as follows:

- if the Secondary Principal was authenticated in the context of security properties of its Primary Principal, the security properties now known for the Primary Principal could affect the Secondary Principal's AUC, for example increasing the authentication level achieved.
- if the Primary Principal's PAC was presented to the Secondary Principal's PA-Server as part of the authorisation for the obtaining of the Secondary Principal's PAC (either explicitly via Declare OperationContext or by being part of the SA within which the GetACT operation was initiated), the security properties now known for the Primary Principal could affect the Secondary Principal's PAC. For example the Secondary Principal may be permitted a higher security clearance if it is known that the Primary Principal itself had a high security clearance.

4.5 Data elements

The APA-Application generates and/or makes use of the following main security information objects:

- Authentication Certificate
- Privilege Attribute Certificate
- Certificate Control Value

- Keying Information
- Generalised User Certificate

Provision is also made in this Standard for the production of other types of certificate. The definition of specific other types is out of scope of this Standard.

4.5.1 Authentication Certificate (AUC)

The AUC is a certificate accepted by an Attribute authority as evidence of a principal's identity for the purpose of providing a PAC. AUCs are created dynamically. An AUC has the following characteristics:

- it is produced by an A-Server to certify the successful completion of an authentication and to identify the principal involved in that authentication,
- it may contain any of the protection fields defined for a certificate (see 4.4.6),
- it contains an attribute of type "Authenticated Identity" for the authenticated principal. This identity is used by PA-Servers for the purpose of supplying PACs to the owner of the AUC,
- it optionally contains an attribute of type "Charging Identifier", used for charging for the use of the system by actions authorised by this AUC,
- it optionally contains an Audit Identity attribute for the principal, against which the principal will be made accountable for its actions, for example in audit trails of its subsequent accesses to protected applications. Other audit-related attributes are also permitted but not defined in this Standard. These attributes will be propagated into all PACs prepared for this principal on this AUC's authority,
- it contains an Authentication Level attribute which indicates the calibre of the authentication that has been performed to obtain this AUC, and therefore the degree of confidence that is to be had in its contents. The PA-Server will subsequently use this field to help decide which Privilege Attributes it is prepared to put into PACs obtained on this AUC's authority,
- it is certified by an Authentication Authority.

4.5.2 Privilege Attribute Certificate (PAC)

The PAC is a certificate that carries the Privilege Attributes of a principal and any other information required to control or monitor the use of these privileges. PACs are created dynamically. A PAC has the following characteristics:

- it is produced by a PA-Server following a valid request for privileges,
- it contains a specification of whether it is a Primary or Secondary Principal, or whether it is for a principal acting as a delegate,
- it contains Privilege Attributes reflecting the access rights of the principal empowered to use this PAC, for example the principal's access identity,
- it optionally contains an attribute of type "Non-repudiation Identity". This identity is used for non-repudiation purposes,
- it optionally contains an attribute of type "Charging Identifier", used for charging for the use of the system by actions authorised by this PAC,
- it optionally contains an Audit Identity attribute for the principal, against which the principal will be made accountable for its actions, for example in audit trails of its subsequent accesses to protected applications. Other audit-related attributes are also permitted, but not defined in this Standard. These are propagated into the PAC from the AUC which authorised the issue of the PAC,
- it may contain any of the protection fields defined for a certificate (see 4.4.6),
- restrictions on the PAC's use can be explicitly specified, either in the certificate itself or linked to it via cryptographic methods,
- it is certified by an Attribute Authority.

4.5.3 Certificate Control Value

The Certificate Control Value (CV) serves to validate the user of an AUC or PAC. It is of particular value when forwarding for use by proxy is required, and when the client does not know precisely which target (e.g. which X-Server application component) will be able to support its request.

The protected certificate contains a Certificate Protection Value (PV) that is a specified one-way function of CV.

Knowledge of CV is used as evidence that the user of a certificate is legitimate. The CV is communicated to the target AEF using a confidentiality service. The use of CV is described in detail in part 2.

4.5.4 Keying Information

Within the context of this Standard, the primary purpose of **Keying Information** is to enable the establishment of a cryptographic key to protect exchanges between APA-Clients and APA-Servers, but it can also be used for the establishment of any shared key under which a PAC is to be protected when passed between two parties. The shared key to be established is called the **basic key.**

The purpose of the basic key is to integrity protect the exchange of security data to prevent it from being undetectably tampered with or misused, and in a few well-defined cases provide confidentiality (key values, CVs and authentication information). This protection needs to be as strong as possible consistent with the security assurance requirements of the system. However there are circumstances in which other data may need to be protected in exchange with a different strength of protection. In particular all of the data exchanged between an initiator and a target, whether it be security data or operational user data, may need to be integrity and/or confidentiality protected. Different keys, possibly using a different algorithm, may then be required for these purposes conformant to local legislative constraints. For example the key or algorithm may need to be weaker if it is to be used for providing confidentiality of all exchanged data. In this Standard such keys are called **dialogue keys**. Dialogue keys are derived from basic keys. They are not directly obtained from Keying Information.

This Standard supports the concept of dialogue keys to provide confidentiality and integrity for dialogues between application components. The use of basic and dialogue keys to protect exchanges between the APA client and the APA-Application is as follows:

- all exchanges are integrity protected with the basic key,
- the confidentiality of Security Information such as passwords or CVs is ensured with the basic key,
- optional confidentiality to globally protect exchanges is ensured with the confidentiality dialogue key,
- the integrity dialogue key is not used by the security services in this Standard (though outside the scope of the Standard it is expected that it would be used for operational dialogues with X-Server application components).

The distinction between dialogue keys and basic keys can be useful for another reason. When the target AEF is a component that is functionally separate from the target application, the basic key can be kept private in the end-system to the AEF, which can control the release of dialogue key(s) to be only if it is satisfied that the incoming access request is valid.

The basic key is used in a variety of situations which may require that different algorithmic information be supplied (on the one hand for example it may be encrypting a single key, and on the other, providing continuous integrity over all security exchanges within a Security Association). Whenever the basic key is used therefore, it is possible to specify an algorithm identifier for its use. In contrast, each dialogue key is used for only one purpose, and therefore the algorithm with which it is to be used can optionally be specified at the time it is established.

Keying Information is structured in two parts: an "**Initiator Key Block**" and an "**Other Key Block**". The former is for the initiator wishing to establish the key. The latter can take one of three forms. Two of these are different kinds of user certificate to help the initiator to perform its own key distribution, the third is a **"target key block",** for use by the target AEF (or the target AEF's KD-Server if different from the initiator's). The parts are designed to cater for a variety of key distribution methods, including symmetric, asymmetric and hybrid schemes. The internal structures of the key blocks and the key distribution mechanisms supported are described in part 2. The initiator and target AEF may share the same KD-Server or may use different KD-Servers.

Initiators wishing to communicate with X-Server application components need not take advantage of the Key Distribution service provided by the APA-Application. Instead they may for example perform their own public key based key distribution with the help of a Directory, but such actions are out of scope of this Standard.

Alternatively they may use the service just to obtain a Generalised User Certificate and associated private key, Directory user certificate and associated private key, or simply trust path information for Certification Authorities.

4.6 Authentication methods

The APA-Application supports a variety of authentication methods and mechanisms through the Authenticate and Continue Authentication operations. These operations are designed to enable multiple exchanges to take place during authentication. The same basic syntactic construct is used for all forms of exchanged authentication information. It is extensible using OBJECT IDENTIFIERS. This Standard does not mandate any particular authentication methods, but specific uses of the authentication information construct for well known ones are proposed in the informative Annex B to part 3. In that Annex, a distinction is made between the authentication of human users and the authentication of application entities, and different methods are proposed for each. Multiple methods may be applied simultaneously.

Among the methods that can be supported within this Standard are:

for human users:

Something Known:

passwords exchanged in clear
passwords encrypted when exchanged
passwords encrypted when exchanged and replay protected
password used to derive a cryptographic key
passwords hashed when exchanged
passwords hashed when exchanged and replay protected
zero knowledge using challenge-response

Something Possessed:

passive token
active token

Immutable Characteristic:

voice print
handwritten signature
fingerprint
retina pattern

Trusted Third Party:

use of private key with directory certificate

Context:

location

for application entities:

Something known:

passwords
symmetric crypto
asymmetric crypto

Context:

application name

4.7 Management and recovery of the APA-Application

The APA-Application conforms to the appropriate standards for Systems Management. Its data and functions are defined in terms of Managed Objects. See 5.6.

All Management and Recovery operations are subject to access control and may require special privileges according to security policy. Management and Recovery operations are controlled by (special) rules recorded in the APA-Application's information store (see part 3 Annex A). This makes the APA-Application self-contained in that it manages its own management operations.

5. Relationship to other Standards and Technical Report

5.1 Relationship to ECMA TR/46, "Security in Open Systems: A Security Framework"

This Standard is based on the concepts of security domains, security authorities and security facilities developed in [ECMA TR/46].

5.2 Relationship to Standard ISO 7498-2, "Security Architecture"

This ECMA Standard permits APA-Application implementors to select implementation profiles whose security does not depend on the provision of either underlying physical communications features or underlying logical communications security services. This approach makes it possible to build secure access control systems without the definition of communications security services of the OSI Reference Model being completed.

5.3 Relationship to Standard ISO/IEC 10181-2, "Authentication Framework"

This Standard is compatible with and makes use of the concepts developed in [ISO/IEC 10181-2], the Authentication Framework. Some types of entities that can in principle be authenticated are mentioned in [ISO/IEC 10181-2]. They are:

- physical entities (e.g. real open systems),
- logical entities (e.g. OSI layer entities),
- human entities.

This Standard is only concerned with the authentication of human users and OSI Applications (including Primary Principals) and is therefore only applicable to a subset of the entities identified in [ISO/IEC 10181-2].

[ISO/IEC 10181-2] also lists the same five different principles used in Authentication as in 4.6

5.4 Relationship to Standard ISO/IEC 9594: 1990 Part 8, "Directory Authentication Framework"

The applications defined by this ECMA Standard complement the conceptual authentication services described by [ISO/IEC 9594] Part 8. Whereas the latter are limited to providing some form of authentication (based on user certificates), this ECMA Standard defines an Authentication application which supports a wide range of authentication methods, which takes an active rather than passive part in the authentication process, and which optionally creates and supplies an Authentication Certificate as evidence of successful authentication. In addition it provides a Privilege Attribute application which supports access control in distributed systems.

However, implementation of the APA-Application may take advantage of the Directory services to store and retrieve information (including user certificates) concerning human users, applications, client systems and server systems.

5.5 Relationship to Standard ISO/IEC 10181-3, "Access Control Framework"

This ECMA Standard is compatible with and makes use of the concepts developed in [ISO/IEC 10181-3]. The correspondence is as follows:

- the Privilege Attributes defined in this Standard are examples of Access Control Information (ACI) as defined in the ACF,
- the PAC defined in this Standard is a particular case of access control certificate,
- the Primary and Secondary Principals defined in this Standard correspond to initiators as defined in [ISO/IEC 10181-3],

- the X-Server application components described in this Standard correspond to targets as used in [ISO/IEC 10181-3],

In terms of the Access Control Framework, The APA-Application provides a trusted source of ACI that is used by initiators to obtain ACI required to gain access to protected targets. Access to the targets is policed by the target AEF, which uses Access Decision Functions which make use of the Privilege Attributes provided by the APA-Application.

This ECMA Standard does not specify how the Access Decision Function (ADF) and Access Enforcement Function (AEF) perform their functions; these are outside the scope of this ECMA Standard, but:

- the ADF is expected to use the Privileges provided by the APA-Application,
- the AEF is assumed to be incorporated in the infrastructure of the application servers.

The application itself may make use of the Privileges in taking its own access control decisions though its own internal AEF and ADF.

5.6 Relationship to ISO/IEC 10164, "Standards for Systems Management"

Management of the APA-Application follows the conventions of ISO Systems Management Standards. Client related attributes and state information are treated as Managed Objects on which management operations can be performed to change the attributes and to initiate actions. The APA-Application also generates event notifications that conform to ISO Management Standards. The managed objects and their attributes related to this ECMA Standard are managed like other non security-related objects according to the conventions of the ISO Systems Management Standards. The same management operations can be applied for managing both the security-related and the non security-related objects. This clause does not deal with management of Security Associations such as monitoring Security Associations (for this see related standards e.g. [ECMA-206]).

5.6.1 Managed Objects

Managed objects in term of ISO standards are the management view of network resources. They are defined as instances of managed object classes. A managed object consists of :

- attributes (which are the properties of the object),
- the value ranges of these attributes,
- the management operations which may be applied to the object, and,
- notifications emitted by the managed object.

Managed objects are created by instantiating managed object classes, i.e. by defining values for their attributes. The managed object classes that represent an APA-Application cover:

- Principal authentication
- Access control information
- Key distribution functions

5.6.2 Management operations on Managed Objects

The ISO management service [ISO/IEC 9595] provides the capability to create or delete managed objects, to get or set or to perform an action on a managed object. For the APA-Application, this includes management functions for:

- principal management
- privilege attribute management
- management of protection values
- security policy management
- management of keys for principals

The management event report service provides the capability for the managed object to send notifications to managers. The APA-Application managed objects include notification for:

- managed object creation,
- managed object modification,
- managed object deletion,
- attribute value change.

5.6.3 Security management functions

The Systems Management Standard [ISO/IEC 10164] identifies the following management functions that may be used to perform security management of the APA-Application:

- The Security Audit Trail Function provides for the collection and review of security-related events for the purpose of monitoring the operation of that portion of the security policy implemented in the Open System, and is specified in [ISO/IEC 10164] Part 8
- The Security Alarm Reporting Function provides event reporting for detecting security attacks and malfunctions, and is specified in [ISO/IEC 10164] Part 7
- The Object Management Function provides the means of managing security objects, and is specified in [ISO/IEC 10164] Part 1
- The State Management Function provides the means of managing the state of the security object, and is specified in [ISO/IEC 10164] Part 2
- The relationship Management Function provides the means of managing the relationship between security objects and other objects, and is specified in [ISO/IEC 10164] Part 3
- The Event Report Management Function provides the ability to establish security audit trails and alarm reporting relationships, and is specified in [ISO/IEC 10164] Part 5
- The Log Control Function provides the ability to configure security audit trails, and is specified in [ISO/IEC 10164] Part 6

Annex A

(Informative)

Changes from the first edition (December 1994)

This second edition of Part 1 of Standard ECMA-219 has been changed from the first edition dated December 1994 in the following respects:

1. Standard ECMA-138 is now withdrawn as it is largely superseded by this Standard, so normative references to Standard ECMA-138 have been removed, and any imported information from it has instead been defined here.

Part 2 - Security information objects

1 Introduction

This part 2 defines the major Security Information Objects used in the APA-Application. Specifically it contains definitions of a Generalised Certificate along with three specific variants: the Authentication Certificate (AUC), the Privilege Attribute Certificate (PAC) and the Generalised User Certificate. An essential component of the AUC or PAC is the Security Attribute. This is also defined here. These definitions are followed by a definition of the Key Set, a package of information provided by the APA-Application on request, containing Keying Information for use by APA-Clients in the provision of protected exchanges.

There are a number of methods by which an AUC and a PAC can be protected from theft and misuse. These are described in detail, along with the External Control Value (ECV) construct which is used with the certificate when one of the protection methods is in use.

Except for the specification of a management port, the three parts of this Standard together provide a complete description of the external interfaces of the APA-Application. Relaying or referral to other servers of the same APA-Application is supported through these interfaces, but the definition of the internal interfaces of such distributed implementations is out of scope of this Standard.

2 Overview

This part defines the major data constructs used in more than one of the operations defined in part 3 of this Standard. In their realisation, IMPLICIT TAGS are to be used throughout. The constructs are:

Generalised Certificates
The different certificates used in the Standard have been constructed as variants of the same basic Generalised Certificate construct, in which fields common to all certificates have been separated out. The different Generalised Certificate types defined here are:

- the Authentication Certificate (AUC), defined in clause 5,
- the Privilege Attribute Certificate (PAC), defined in clause 6,
- the Generalised User Certificate (GUC), defined in clause 8

External Control Values
Some of the protection methods, which are defined in clause 7 for preventing AUCs and PACs from being stolen or misused, require additional information, to be provided along with the certificate when it is presented for access. This construct contains this information. It is defined in clause 9.

Keying Information
The key distribution functions defined in part 3 operate with these structures, which are designed to be common to many potential key distribution schemes. They are defined in clause 10. The Keying Information contains internal fields whose structures are not defined here, but are expected to be fixed when choice and definition of a particular scheme has been made. Such a definition is out of scope of this Standard.

The AUC and PAC both contain attributes relating to the identity of the principal for whom the certificate has been issued. The concept of identity is a complex one, and Annex A describes the approach taken in this Standard.

Annex B contains the formal definition of all of the ASN.1 defined in this part. The constructs are sorted into alphabetical order.

3 GeneralisedCertificate

A GeneralisedCertificate is composed of three main structural components :

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

The "specificContents" fields are different for different types, and contain a type identifier to indicate the type. In this Standard three types are defined in detail: the Authentication Certificate (AUC), the Privilege Attribute Certificate (PAC), and the Generalised User Certificate (GUC).

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

"commonContents" and "checkValue" contain fields common to all certificates. "commonContents" and "specificContents" constitute the "certificateBody" which may be encrypted.

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

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

The next three clauses describe these three main structural components of the GeneralisedCertificate.

3.1 Common Contents fields

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

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

NOTE
In the imported definition of AlgorithmIdentifier, ISO currently permits both a hash and a cryptographic algorithm to be specified. If this is done, they must appear in the algId field. The hashAlgId field is present for those cases where a separate hash algorithm specification is required.

```
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}}

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

comConFieldsSyntaxVersion
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.

NOTE
Identifier is a general syntax which is used for all fields containing names or identifiers of various kinds. It can take any of the forms listed: object identifier, Directory DN or RDN, a printable name, an octet string, an integer value, a bit string or a paired value consisting of readable and unique components.

serialNumber
The serial number of the certificate 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.

3.2 Specific Certificate Contents

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

SpecificContents

Different types of certificate are identified by different choices as follows:

- **auc** identifies an Authentication Certificate,
- **pac** identifies a Privilege Attribute Certificate that can be used by its owner either when acting as an original initiator or acting by proxy (i.e. for use in the Trace Pointer PAC protection method described in clause 7) for another initiator,
- **proxyOnlyPAC** identifies a Privilege Attribute Certificate that can be used by its owner only when acting by proxy for another initiator,
- **initiatorOnlyPAC** identifies a Privilege Attribute Certificate that can be used by its owner only when acting as an original initiator,
- **gucType** identifies a Generalised User Certificate,
- **otherType** identifies a type of certificate not specifically defined in this Standard.

3.3 Check value

The certificate can be protected either by a signature or a symmetric seal. It is the certificateBody construct that is protected by the check value.

If a signature is used, it 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.

If a seal is used, the seal value is accompanied by information indicating the method and key by which the seal was generated.

If the certificate body is encrypted, the encryption algorithm used is specified in the seal, for symmetric encryption, or in the signature for asymmetric encryption. Otherwise, the algorithms used to generate the check value are specified in the common contents part of the certificate.

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

```
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}

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

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

asymmetricAlgId
Only present if the certificate body is encrypted, then it is a duplication of the algId value in "commonContents".

hashAlgId

Only present if the certificate body is encrypted, then it is a duplication of the hashAlgId value in "commonContents".

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].

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

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

sealValue
The value of the seal. It is the result of a symmetric encryption of a hash value of a certificateBody

symmetricAlgId
Only present if the certificate body is encrypted, then it is a duplication of the algId value in "commonContents".

hashAlgId
Only present if the certificate body is encrypted, then it is a duplication of the hashAlgId value in "commonContents".

targetName
This field identifies the target AEF 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.

3.4 Certificate Identity

Certificate identity is a named concatenation of the three named field types defined in CommonContents (see 3.1) which together uniquely identify a certificate. This construct is used in many of the operations of part 3.

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

4 Security attributes

The SecurityAttribute is a basic construct used in all certificate types defined in this Standard.

```
SecurityAttribute ::= SEQUENCE{
      attributeType    Identifier,
      attributeValue   SET OF SEQUENCE{
                             definingAuthority   [0] Identifier          OPTIONAL,
                             securityValue       [1] SecurityValue} }
      --  NOTE: SecurityAttribute is not tagged, for compatibility with the Directory Standard.

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

NOTE
Only one set member is permitted in attributeValue. Multivalue attributes are effected in the securityValue field, where the "SEQUENCE OF" construct can be used. This restriction avoids any certificate hash value ambiguity that may occur with indeterminacy in the representation of the order of SET members. At the same time, by including "SET OF" in the syntax, the Standard enables security attributes to be stored as normal in a Directory whenever the choice made within Identifier is OBJECT IDENTIFIER. If other choices are made, encapsulation in an outer Directory Attribute structure would first be necessary.

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
The authority responsible for the definition of the semantics of the value of the security attribute. This optional field of the attributeValue can be used to resolve potential value clashes.

NOTE
This is not to be confused with the Attribute Authority responsible for controlling which principals are permitted to use which attribute values in PACs - though under some security policies they may be the same.

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.

5 Authentication Certificate (AUC)

The syntax of the AUC contents is:

```
AUCSpecificContents ::= SEQUENCE {
    aucSyntaxVersion      [0]   INTEGER   {   version1 (1)}        DEFAULT 1,
    protectionMethods     [1]   SEQUENCE OF MethodGroup            OPTIONAL,
    aucType               [2]   ENUMERATED{  primaryPrincipal           (1),
                                             secondaryPrincipal         (2)},
    authenticationLevel   [3]   INTEGER                            OPTIONAL,
    aucAtts               [4]   SEQUENCE OF SecurityAttribute,
    timePeriods           [5]   TimePeriods                        OPTIONAL}

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

aucSyntaxVersion
Syntax version of the AUC

protectionMethods
A sequence of optional groups of Method fields used to protect the certificate from being stolen or misused. For a full description see clause 7

aucType
Indicates whether the AUC owner is a Primary Principal or a Secondary Principal.

authenticationLevel
Level of the authentication. It indicates the level of confidence in the authentication method used.

aucAtts
Security attributes that are put in the AUC for use by the PA-Server. It always contains the authenticated identity and optionally other attributes, among which are:

- Audit identity,
- Charging identifier.

NOTE
The name "Charging Identifier" is used in preference to "Charging Identity" to distinguish it from the other identities which are all identities of the certificate user. "Charging Identifier" names, and therefore identifies a different entity, typically an account.

timePeriods
This field adds further time restrictions to the validity field of the commonContents. Either startTime or endTime can be optional. The TimePeriods control is passed if the time now is within any of the sequence periods, or if there is a period with a start before now and no endTime, or there is a period with an end after now and no startTime.

6 Privilege Attribute Certificate (PAC)

The syntax of the PAC contents is:

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

For MethodGroup and its sub-fields see clause 7.

```
PrivilegeAttribute ::= SecurityAttribute

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 in this Standard.
    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
```

pacxSyntaxVersion
Syntax version of the PAC.

pacHistory
This field is for use in Inter-domain Services. It is used whenever an Inter-domain Service creates and signs a new PAC based on a PAC received from another domain, the certificateIdentifier of the incoming PAC is added to this field, thus providing a history of the PACs which led to the creation of this PAC.

protectionMethods
A sequence of optional groups of Method fields used to protect the certificate from being stolen or misused. For a full description see clause 7.

traceLink
Indicates the security attributes of an application performing proxy. This may simply be the name of the application or the name of one of the groups to which the application belongs. It is only used in conjunction with the "Trace Pointer" protection method (see 7.6). When the application is not acting as a delegate (e.g. it is the first application of a chain) this field may have a null value.

pacType
Indicates whether the privileges contained in the PAC are those of a Primary Principal, of a Secondary Principal tempered by the privileges of a Primary Principal (the privileges of the Secondary Principal which appear in the PAC are a subset of the whole set of the privileges of the Secondary Principal, taking into consideration the privileges of the Primary Principal - or the lack of knowledge of them) or of a Secondary Principal (the privileges of the Secondary Principal have not been affected by any privileges of the Primary Principal) .

privileges
Privilege Attributes of the principal.

restrictions
This field enables the original owner of the PAC to impose constraints on the operations for which it is valid. A restriction is a bit string which can be either included in the PAC or external to it. A restriction can be targeted to one or more application components. Two types of restriction are supported:

- Mandatory: If a target application component to which the restriction applies cannot understand the bit string defining the restriction, access should not be granted,
- Optional: If a target application component to which the restriction applies cannot understand the bit string, it is expected to ignore it.

Three different ways of expressing the restrictions are supported :

> **hashedExternal**
> The restriction is carried outside the PAC but is bound to it by use of irreversible cryptographic techniques :
>
> The APA-Application inserts into the PAC the result of a hash function applied to the external restrictions, along with the hash algorithm identifier.
>
> **signedExternal**
> The restriction is carried outside the PAC but is bound to it by use of asymmetric cryptographic techniques :
>
> Either the client asks the APA-Application to generate a private key for it, or the client sends the APA-Application a public key corresponding to a private key it already knows, along with information about the way in which the key is to be used. The APA-Application inserts the public key received (or that it generated) along with usage information in the PAC. If the APA-Application has generated a private key for the client, this key is returned encrypted under the basic key in an ECV construct (see clause 9).
>
> When the client now sends the PAC to a target system, the target application component will expect constraints associated with that PAC to have been signed by the private key corresponding to the public key in the PAC. This Standard does not dictate the details of these signing operations.
>
> **certExternal**
> As above, but a reference to a certification authority and user certificate identifying the public key is present instead of the actual key. See 3.4 for the syntax.
>
> **included**
> The restriction is directly included in "restrictions" field of the certificate.

miscellaneousAtts
Security attributes which are neither privileges attributes nor restrictions attributes. In a PAC, this includes identity attributes such as :

– Non-repudiation Identity

– Audit Identity

– Charging Identifier

NOTES

1) *The name "Charging Identifier" is used in preference to "Charging Identity" to distinguish it from the other identities which are all identities of the certificate user. "Charging Identifier" names, and therefore identifies a different entity, typically an account.*
2) *This Standard does not preclude the authenticated identity to be one of the miscellaneous attributes. However, when it is present, it must not be used for access control purposes.*
3) *Access identity security attributes (as described in Annex A), if present in the PAC, appear in the privileges field.*

timePeriods
This field adds further time restrictions to the validity field of the commonContents. Either startTime or endTime can be optional. The TimePeriods control is passed if the time now is within any of the sequence periods, or if there is a period with a start before now and no endTime, or there is a period with an end after now and no startTime.

7 Protection methods

```
MethodGroup ::= SEQUENCE OF Method

Method ::= SEQUENCE{
    methodId            [0]   MethodId,
    methodParams        [1]   SEQUENCE OF Mparm              OPTIONAL}

MethodId ::= CHOICE{
    predefinedMethod [0] ENUMERATED {  controlProtectionValues        (1),
                                         ppQualification              (2),
                                         targetQualification          (3),
                                         delegateTargetQualification  (4),
                                         delegateQualification        (5),
                                         tracePointer                 (6),
                                         initiatorQualification       (7),
                                         count                        (8),
                                         checkBack                    (9),
                                         nestedChain                  (10)},
    otherMethod      [1] OBJECT IDENTIFIER}

Mparm ::= CHOICE{
    pValue               [0]   PValue,
    securityAttribute    [1]   SecurityAttribute,
    integerValue         [2]   INTEGER,
    octetValue           [3]   OCTET STRING,
    bitStringValue       [4]   BIT STRING,
    nestedCertificate    [5]   CertandECV,
    printableStringValue [6]   PrintableString,
    certificateId        [7]   CertificateId,-- defined in 3.4
    otherValue           [8]   SEQUENCE{
                                   paramType  [0] OBJECT IDENTIFIER,
                                   paramValue [1] ANY -- defined by paramType}
```

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

CertandECV ::=     SEQUENCE {
    certificate            [0]     GeneralisedCertificate,
    ecv                    [1]     ECV,                        OPTIONAL}
                -- ECV is defined in clause 9
```

methodId
Identifies a protection method. Several methods are defined in this Standard. Alternative methods are also provided for by means of the "otherMethod" OBJECT IDENTIFIER. 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 as described below. It is the responsibility of standards definers who define alternative methods to specify the syntax choices available for their methods' methodParams.

methodParams
Parameters for a protection method. The semantics of each protection method is described in 7.1 and following clauses below. In each case, the syntax choice of the mParm field is defined. Unless otherwise stated, a protection method applies equally to AUCs and PACs. If the certificate is an AUC, references to target X-Server application components should be interpreted to mean specifically PA-Server application components.

7.1 "Control/Protection Values" protection method

The MethodId for this is: controlProtectionValues

This method protects the certificate from being stolen, at the same time controlling its acceptance for use by proxy to defined groups of target X-Server application components. The syntax choice of Mparm for this method is:

pValue.

This scheme uses a certificate Protection Value (PV) to provide a method of controlling the forwarding of a certificate for use by proxy, without the user of the certificate needing to know the precise identity of the final X-Server application component. It prevents a line-tapper from stealing the certificate for his own use, but does not demand that the certificate be encrypted. A maximum of one PV method is permitted in each method group. Security attributes can be associated with the PV by including appropriate protection methods in the PV's method group.

More than one method group can be specified, each containing a PV.

Also associated with each PV is a certificate Control Value (CV) external to the certificate. The PV is the result of a one-way function applied to the corresponding CV. When the certificate is offered to an X-Server application component, it is proof of knowledge of this CV which causes the certificate to be accepted (subject to the other controls in the PV's method group). Unless otherwise specified, the one-way function that is used for this method is MD5.

Either the CVs are generated and PVs calculated by the APA-Application for the client on request for each occurrence of the method in the certificate, or the client itself generates the CVs and specifies the PVs to the APA-Application. The PVs are returned in the certificate; the corresponding CVs, if they have been generated by the APA-Application, are returned along with the certificate, encrypted under the key used to protect the certificate request. They are returned in the ECV construct described in clause 9.

When the client now sends the certificate to a target AEF, one or more CVs are also sent encrypted under the basic key used to communicate with the target AEF. They are sent in the ECV construct. Not all of the CVs associated with the certificate need always be sent. The target AEF 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 certificate is acceptable under this method group.

The target AEF now knows the value of the CVs that have been sent to it, and can make available this value to the application or to the infrastructure supporting that application so that it can forward the certificate for use with another X-Server application component. By including target qualification controls in the method group, proxy can be confined to groups of X-Server application components, for example all of the servers in a distributed service.

This form of certificate protection, in which only the CVs are required to be encrypted, is useful for systems where user data encryption is not allowed by law or not accepted practice.

7.2 "Primary Principal Qualification" protection 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. 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. It can also be used to permit proxy, when the required attributes of the proxy application Primary Principals are precisely known. The method does not limit the number of X-Server application components at which a certificate might be accepted. 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 attributes are inserted in the certificate by the APA-Application according to security policy. They may be requested by the client. They are attributes that must be possessed by any Primary Principal from which this certificate is to be validly used. When a target AEF receives such a certificate, it is responsible for comparing the attributes found in the certificate with the attributes that it knows (by other means described below) are associated with the initiating Primary Principal. Only if the initiator has all of the ppQualification attributes in the certificate, is the certificate accepted under this method.

There are two main ways by which a target AEF can know the attributes associated with a Primary Principal.

The first is by some form of peer-entity authentication which may have taken place in a manner outside the scope of this Standard.

The second is by the use of Security Associations or Contexts established via Key Sets as defined by this Standard. Whenever an initiator sends a certificate to a target AEF, it does so under the protection of a key established by use of the targetKeyBlock constructs defined in clause 10. The targetPart of the targetKeyBlock defined there optionally contains a construct which includes a sequence of Primary Principal security attributes. Alternatively when the initiator is performing its own key establishment, the Generalised User Certificate that it uses contains these values, and to prove that the user certificate is its own, the initiator is expected to sign the key it sends to the target AEF (as well as protecting it by encrypting it under the public key of the target AEF).

The attribute values are placed in the targetKeyBlock or Generalised User Certificate by the trusted server (the KD-Server) which created it. They are security attributes of the initiating Primary Principal that requested the key block or certificate, as told to the KD-Server by the APA-Application intermediary. It is these attributes that the target AEF compares with the security attributes in the Mparm field.

A special case exists for a client requesting a certificate from the APA-Application when the client Primary Principal's attributes are not known to the APA-Application (for example the client may be in a dial-up workstation). For this case, the client asks for the APA-Application to generate an arbitrary unique value as its Primary Principal's single attribute. This is placed in the certificate as a ppQualification method mParm value. When the client subsequently requests a Key Set from the APA-Application, the APA-Application ensures that the same unique value is inserted as an attribute in the targetKeyBlock, or Generalised User Certificate produced by the APA-Application for the client.

7.3 "Target Qualification" protection method

The MethodId for this is: targetQualification.

This method protects the certificate from misuse by allowing its use only by a nominated set of X-Server application components and at the same time preventing it from being forwarded by them.

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 attributes are inserted in the certificate by the APA-Application according to security policy. They may be requested by the client.

Target AEFs receiving such Certificates will compare the values found in the Certificate with the attributes of the X-Server application component. If the X-Server application component 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 X-Server application components.

7.4 "Delegate/Target Qualification" protection method

The MethodId for this is: delegateTargetQualification

This method protects the certificate from misuse by confining its validity to a set of X-Server application components.

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 attributes are inserted in the certificate by the APA-Application according to security policy. They may be requested by the client.

The target AEF makes the comparisons described in 7.3, but if the checks are passed, the nominated X-Server application component is acceptable as both an accessible target and as a delegate. The attributes carried by the certificate are valid at the X-Server application component for authentication or access control purposes and the target AEF will allow the use of the certificate for access to further X-Server application components.

7.5 "Delegate Qualification" protection method

The MethodId for this is: delegateQualification

This method protects the certificate from misuse by allowing it to be forwarded but at the same time preventing its use at the delegate.

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 attributes are inserted in the certificate by the APA-Application according to security policy. They may be requested by the client.

The target AEF makes the comparisons described in 7.3, but if the checks are passed, the nominated X-Server application component is accepted only as a Delegate. None of the attributes carried by the certificate are valid at the X-Server application component for authentication (in the case of an AUC) or for the control of access to that component (in the case of a PAC), but nevertheless the target AEF will allow the use of the certificate for access to further X-Server application components.

7.6 "Trace Pointer" protection method

The MethodId for this is : tracePointer.

This method is only applicable to PACs. It enables the construction of a chain of PACs so that an X-Server application component at any stage on the delegation path can trace the route followed by a given operation through previous delegates back to the original initiator.

Two values are associated with the tracePointer method:

– The first value is called the forwardPointer which points to the traceLink field of the next valid PAC in the chain (see clause 6 for a description of this field). It does this by specifying security attribute(s), one of which the traceLink field of the next PAC must contain. This may simply be the name of the application owning that PAC, or the name of one of the groups the application belongs to. Thus, if traceLink is not present in a PAC, it must be the first in any chain.

– The second value is called the backwardPointer. Its presence is optional and points to the PAC identifier of the previous PAC. It is needed only when multiple PACs using this protection method are sent by an intermediate application towards the same next X-Server application component.

The syntax choice of Mparm for this method is:

securityAttribute	for the forwardPointer, and
certificateId	for the backwardPointer.

In the sequence of Mparm construct, at most one certficateId, but one or more security attributes can be present.

The attributes are inserted in the certificate by the APA-Application according to security policy. They may be requested by the client.

The same PAC can be issued at different times with different delegation routes being permitted. This is done by pairing different tracePointer values with CV/PV methods in a number of method groups. Choice of CV sent with the PAC effectively activates the method group in which the corresponding PV is to be found, and consequently activates the associated tracePointer values.

A target AEF receiving the chain of PACs will check that all chain links match correctly starting from the first PAC to the last (for a full description of the PAC validation logic see 7.11).

7.7 "Initiator Qualification" protection method

The MethodId for this is: initiatorQualification

This method provides weak protection for the certificate from being used by an invalid initiator. The method parameter specifies security attributes which the initiator can state without verification. An example value would be a network address as used in Kerberos.

The target AEF compares this value with corresponding attributes supplied externally by means not defined in this Standard.

The syntax choice of Mparm for this method is:

securityAttribute

7.8 "Count" protection method

The MethodId for this is: count

This method protects a certificate from misuse by limiting the number of times it can be validly presented to targets.

The syntax choice of Mparm for this method is:

integerValue.

The Count value is inserted in the certificate by the APA-Application according to security policy. It may be requested by the client.

When a target AEF for an X-Server application component satisfying any other controls that may pertain in the method group containing the count first receives the certificate from an initiator, it registers the value of the count against that application component. Subsequent receipts of the same certificate cause the count to be decremented. The target will not accept as valid any certificate that has been presented more than the number of times specified in the count.

7.9 "Check Back" protection method

The MethodId for this is: checkBack

This method protects a certificate from misuse by requesting a target AEF receiving such a certificate to check back with the original issuing APA-Application to ensure that the certificate is still valid. The method provides for instant revocation of a certificate.

No methodParams construct is used for this method.

Only one occurrence of the method is permitted in any one method group. The method is inserted in the certificate by the APA-Application according to security policy. It may be requested by the client. When a target AEF receives such a certificate, it is expected to pass it or identify it to the originating APA-Application in a CheckAUC or CheckPAC operation according to the type of certificate (see part 3).

7.10 "Nested Chain" protection method

The MethodId for this is: nestedChain

This method is only applicable to PACs. The method supports delegation with PAC chaining by nesting PACs within other PACs, step by step, so that a single PAC construct can be passed on each delegation leg for verification. Each receiving target AEF needs verify only one signature: the signature on the outer PAC. It therefore needs to recognise only the last Attribute Authority in the chain, and the X-Server access protocol needs only to be capable of carrying one PAC construct.

Two syntax choices are associated with the nestedChain method:

securityAttribute

nestedCertificate

– The first value is a forward pointer which points to the traceLink field of the next outer PAC in the nest (see clause 6 for a description of this field). The forward pointer specifies security attribute(s), one of which must be contained in the traceLink field of the next outer PAC (in the same manner as in the Trace Pointer method). This may simply be the name of the application owning that PAC, the name of its Primary Principal, or the name of one of the groups the application belongs to. Thus, if traceLink is not present in a PAC, that PAC must be the innermost in any nesting.

– The second value is used to contain the actual embedded PACs.

NOTE
The ECV component of this field is not used within the PAC, but is present to allow CVs to be presented to the APA-Application in the certificateControls field of the ticketRequirements argument in Get ACT or Refine PAC operations (see below, and see part 3 for a description of these operations).

When a delegate's AEF receives an incoming PAC which includes this method, it must use the APA-Application to construct a new PAC for forwarding. It sends its APA-Application the incoming PAC, with its CVs as part of one of the certificate controls in the ticketRequirements argument in a Get ACT or Refine PAC operation, and the APA-Application embeds it, modified as described below, in the new returned delegate's PAC, inside the method parameter. Before doing this, the APA-Application is expected to verify the check value of the existing PAC, check the CVs presented with that PAC, and verify the correctness of the forward pointer part of the method parameters in the incoming PAC. This forward pointer must match the traceLink value in the new delegate's PAC as described above. None of the incoming PAC's CVs are returned with the new PAC. All PV methods are deleted from their method groups, which may as a result become empty, but remain present. Method groups determined to be invalid on the basis of the checks made are deleted in their entirety. If the forwardPointer check fails, the embedding operation is rejected. Finally the Check Value field in the embedded PAC is set to "unprotected".

Other control parameters in the incoming PAC could optionally also be verified by the APA-Application for early detection of authorisation failures, but without any effect on the PAC content and without removing the obligation on the final AEF to perform these checks.

The new PAC representing the extended chain is returned to the AEF which makes it available to the delegate for onward use with a further X-Server.

The PA-Server that creates each new PAC is responsible for the representation of the whole chain up to that point, and each AEF needs to trust the PA-Server it uses, to make this representation according to security policy. In a multi-hop chain, a transitive trust relationship builds up between PA authorities.

To initiate the use of this method for the whole delegation route, the original initiator's PAC is caused to contain a nestedChain method in a valid method group with a method parameter containing a forward pointer set to a meaningful value, but without a CertandECV field.

As part of the Nested Chain validation process, AEFs are expected to validate each nested PAC in incoming PACs as normal, including time periods and restrictions, and according to the method groups remaining. However, although all of the contained PACs' forwardPointers are still present, they have all been validated en route by the embedding APA-Application(s), so only the final AEF (which does not use the APA-Application to embed the received PAC) needs validate that the forward pointer in the outermost PAC points at the final target X-Server application component.

7.11 Combining the methods

Certificates are unlikely to contain complex combinations of many method types with multiple occurrences of each type. It is expected that one occurrence of each of one or two method types in one method group would be normal. However the general rule for validating a certificate when received by a target AEF is as follows:

If the certificate is properly signed by an authority recognised by this target AEF and is within the valid time periods, then if a PAC is the first in a chain of PACs being presented it must not be a proxyOnlyPAC. If it is not the first in such a chain (i.e. it has been added by a delegate under the Trace Pointer protection method - see 7.6), it must not be an initiatorOnlyPAC. If these checks are passed (or not required), then the protection methods in each group are tested in turn as described below until one of the groups is passed or the certificate is declared invalid.

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

the certificate is for the nominated target X-Server application component under any target, delegateTarget or delegate qualification method in this group

and

it does not violate any count or check-back or, in the case of a PAC, nested chain checks in the group

and

tests on any ppQualification, initiatorQualification or controlProtectionValues method in the group succeeds. If more than one of these is present, if initiatorQualification is one of them, it is ignored, and at least one of the others must be passed. If only one is present it must be passed. If none of them is present this last check is not required.

Empty method groups may appear in PACs as a result of APA-Application actions in support of the Nested Chain protection method (see 7.10).

If the group that has been passed only validates the certificate for its recipient application component being a delegate, then further groups are checked to see if the recipient is also valid as a delegate/target or target. If, following these additional checks, a recipient is still valid only as a delegate, none of the attributes carried by the certificate are accepted as valid at the recipient for authentication or access control purposes.

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 or delegate/target. If, following these additional checks, a recipient is still valid only as a target, the target AEF is responsible for preventing its use for access to further target X-Server application components.

If the certificate is a PAC and is presented as part of, but not the last of a traced delegation chain then at least one method group valid for this recipient as a delegate or delegateTarget, and containing a tracePointer, must be present. One of these tracePointers must have a forwardPointer value corresponding to the traceLink field of the next PAC in the chain. If this forwardPointer is paired with a backwardPointer, the backwardPointer value must also correspond to the issuerDomain, issuerIdentity and serialNumber fields of the commonContents part of the preceding PAC in the chain. There must be such a PAC.

If the certificate is a PAC and is presented as the last of a traced delegation chain then, if present in an otherwise valid method group, the backwardPointer value of any tracePointer method must correspond to the issuerDomain, issuerIdentity and serialNumber fields of the commonContents part of the previous PAC in the chain.

If the certificate is a PAC and is presented as part of a nested chain, then the checks described at the end of 7.10 must be passed.

8 Generalised User Certificate (GUC)

A (GUC) signed by the APA-Application and containing a public key which corresponds to a temporary private key also generated and supplied by the APA-Application. The private key may be used for establishing a Security Association with target AEFs (see clause 10). It may also be used for signing external restrictions (see the description of signedExternal in clause 6).

The certificate also contains security attributes associated with the owner of the private key; in the context of this Standard this is the APA-Client acting as part of a Primary Principal. In cases where a GUC is used, it is sent to target AEFs as part of the Open SA operation. If a certificate protected under the Primary Principal Qualification method is subsequently received from this client, these attributes will be used by the target AEF to compare with the Primary Principal qualifier attributes in the certificate. This is more fully described in 7.2.

The syntax of a user certificate contents is:

```
GUCSpecificContents ::= SEQUENCE{
                                    attributes    [0]    SEQUENCE OF SecurityAttribute,
                                    publicKey     [1]    BIT STRING}
```

9 External control values construct

Whenever the protection controlProtectionValues method is in place, when an AUC or 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. Also, when such a certificate is being issued to a requesting client, the CV values it will need in order to use that certificate may need to be returned with it.

Similarly, private keys may need to be returned from the APA-Application for use with external signedRestrictions. In order to cater for these and for any future protection methods which may require External Control Values, a standard construct is used for them as follows:

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

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

crypAlgIdentifier
This specifies the encryption algorithm of the control values.

cValues
An ECV construct can contain either an encrypted list of control values in the **encryptedCvalueList** field, or a list of individually control values in **individualCvalues**.

If the encryptedCvalueList choice is made, the whole list is encrypted in bulk, but the in-clear contents of this field are expected to have the syntax CValues. If the individualCvalues choice is made, values are individually encrypted in the **value** fields of the list. Encryption is always done under the basic key protecting the operation.

In the case of the controlProtectionValues method, **value** is a CV, and **index** is then the index of the method occurrence in the certificate, starting at 1.

In the case of external signed external restrictions, **value** is a private key, and **index** is then set to zero.

10 Specification of Keying Information

Keying Information is used in the establishment of **basic keys** as described in part 1 of this Standard. Keying Information can be obtained from any of the three operational ports of the APA-Application, though the KD-Service is the real provider. When the A- and PA-Service ports are used the A- and PA-Services are acting as trusted intermediaries.

Dialogue keys can be derived from basic keys. The information to make that derivation is transmitted with the information used to establish the basic key, as described below in 10.2.2.

10.1 Configurations Supported

Keying Information for basic key establishment is structured to be able to support a variety of key distribution protocols. It is provided in a **Key Set**.

Keying Information may be either built locally or obtained from a KD-Server. The first option, illustrated in 1) below, is only possible when public keys are used, The second, illustrated in 2) below, is always possible.

1) When the initiator is prepared to build keys itself and when the target AEF long term key is a public key, a Basic Key may be established directly without using a KD-Server. This implies that Key Certificates (i.e. [ISO/IEC 9594-8] user certificates) and revocation lists are being used and that the initiator is able to access that information directly or indirectly. No Keying Information is required at all for this class of initiator. However an initiator may want to obtain from a KD-Server either or both of two kinds of information:

   - trust conditions saying which CAs can be trusted for what. This is returned in the trustInformation construct of the GetKI operation result. It is described in part 3,
   - either a temporary asymmetric key pair or a permanent asymmetric key pair (if the initiator has one). The asymmetric key pair can be used to provide basic key constructs which can support the "Primary Principal Qualification" method for non-proxiable PAC control. This is returned in the Key Set. There are two pieces of information:
     - an **initiatorKeyBlock** from which the private key can be obtained by the initiator
     - an **otherKeyBlock** containing a public key certificate, taking the form either of an [ISO/IEC 9594-8] user certificate or a Generalised User Certificate.

2) When the initiator obtains a basic key from the KD-Server, it receives back two pieces of information in the Key Set:

   - an **initiatorKeyBlock** which allows the initiator to recover the value of the basic key,
   - an **otherKeyBlock** taking the form of a **targetKeyBlock** valid for the specified target AEF.

   The initiator does not need to understand the content of the Target Key Block but simply supplies it for forwarding to the target AEF. This gives some flexibility over the different schemes which may be used while keeping the initiator's code unchanged.

In some protocols, the initiator and the target AEF share a single KD-Server; in others they use two separate KD-Servers. Separate KD-Servers are needed for inter-domain key distribution.

Long term keys (either for symmetric or asymmetric cryptography) are needed for key distribution . This Standard supports the construction of data structures based on the following underlying long term key configurations between the target AEF, the target AEF's KD-Server and the initiator's KD-Server:

1. The initiator shares a KD-Server with the target AEF. The target AEF shares a symmetric long term key with the KD-Server,
2. The initiator shares a KD-Server with the target AEF. The target AEF and the KD-Server possess private keys,
3. The initiator and target AEF have different KD-Servers, the two KD-Servers share a symmetric long term key,
4. The initiator and target AEF have different KD-Servers, the two KD-Servers possess private keys

In the first two configurations there is no direct communication between the target AEF and the shared KD-Server. In last two configurations, the target AEF performs exchanges with its own KD-Server.

A number of the key distribution protocols permit the interim establishment of cached Dynamically Established Secret Keys (DESKs) shared between the initiator's KD-Server and the target AEF's KD-Server (if different). The use of DESKs provides performance benefits in subsequent key distribution operations.

Examples of DESK variants of the above configurations are given below:

5. A refinement of Configuration 2 above (Single KD-Server, the target AEF and KD-Server each possessing a private key). The private keys are used to establish a DESK which can subsequently be used as in Configuration 1 to establish keys between the initiator and target AEF,

6. A refinement of Configuration 4 above (Separate KD-Servers each possessing a private key). The private keys are used to establish a DESK which can subsequently be used as in Configuration 3.

Public Key based configurations are:

7. [ISO/IEC 9594-8] style, using a long term private key and an [ISO/IEC 9594-8] user certificate supplied by the KD-Server,

8. [ISO/IEC 9594-8] style as above, but using a Generalised User Certificate issued by the KD-Server acting as a Certification Authority.

10.2 General Description

The **KeySet** is structured into two parts: the **initiatorKeyBlock** from which the initiator obtains a key directly, and the **otherKeyBlock**.

If the initiator is using its KD-Server simply to provide a private key and a public key certificate (to do its own key distribution along the lines of [ISO/IEC 9594-8]), the initiatorKeyBlock contains the private key and the otherKeyBlock contains the public key certificate (with the **ucECMA** or **ucDirectory** choices being made).

Otherwise, the otherKeyBlock takes the form: **targetKeyBlock**, which provides information for the target AEF, or the target AEF's KD-Server, to use. Once the initiator has determined that this type of otherKeyBlock has been provided, it need not look further into it, but can simply pass it on to the target AEF without needing to know which of the forms of key distribution is in use. The initiatorKeyBlock is unaffected by these choices.

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 containing the name of the initiator’s KD-Server **initiatorKDSname**,
- a part which, if present, the target AEF needs to pass on to its KD-Server (**targetKDSPart** - will be present only when the target AEF's KD-Server is different from the initiator's),
- a part which, if present, can be used directly by the target AEF (**targetPart**), but after processing of the **targetKDSPart** when it exists.

When a target AEF using a separate KD-Server receives the targetKeyBlock, it first checks whether it supports the key distribution scheme indicated in kdsSchemeOID. Then three 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 certificate protection under the Primary Principal Qualification method (see 7.2).

2) Only the targetKDSPart is present. The target AEF uses the ProcessKI operation (described in part 3) to forward kdSchemeOID, initiatorKDSname, and targetKDSPart to its KD-Server. 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 certificate protection purposes. Expiry information can optionally be present in the targetKDSPart.

3) Both the targetPart and the targetKDSpart are present. The target AEF uses the ProcessKI operation to forward initiatorKDSname and targetKDSpart to its KD-Server. In return it receives Keying Information which used in conjunction with the targetPart enables it to calculate the basic key. Once again, 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 certificate protection under the Primary Principal Qualification method.

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

10.2.1 Keying Information Syntax

```
KeySet ::= SEQUENCE {
        initiatorKeyBlock           [0]     InitiatorKeyBlock,
        otherKeyBlock               [1]     OtherKeyBlock}

InitiatorKeyBlock ::= SEQUENCE {
        keyConstructionData         [0]     KeyConstructionData,
        keyUseData                  [1]     KeyUseData                     OPTIONAL,
        validity                    [2]     Validity                       OPTIONAL}
                                                    -- defined in 3.1

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

MethodParameters ::= CHOICE {
        oneWay                      [0]     OneWayParameters,
        encrypted                   [1]     SEQUENCE {
                encryptedKeyValue           [0]     BIT STRING,
                keyIdentifier               [1]     INTEGER                        OPTIONAL,
                symmetricAlgId              [2]     AlgorithmIdentifier OPTIONAL},
        other                       [2]     ANY}          -- defined by constructionMethodId

OneWayParameters ::= SEQUENCE {
        seed                    [0]     BIT STRING,
        keySize                 [1]     INTEGER                            OPTIONAL,
        hashAlgId               [2]     AlgorithmIdentifier                OPTIONAL,
        sourceKeyId             [3]     INTEGER                            OPTIONAL }

KeyUseData ::= SEQUENCE OF TargetName

TargetName ::= Identifier                      -- Identifier is defined in 3.1

OtherKeyBlock ::= CHOICE {
        ucECMA              [0] GeneralisedCertificate,
        ucDirectory         [1] Certificate,        -- imported from the Directory Standard
        targetKeyBlock      [2] TargetKeyBlock}

TargetKeyBlock ::= SEQUENCE {
        initiatorKDSname        [0]     Identifier              OPTIONAL,
        kdSchemeOID             [2]     OBJECT IDENTIFIER,
        targetKDSpart           [3]     ANY                     OPTIONAL,
                                                    -- depending on kdSchemeOID
        targetPart              [4]     ANY                     OPTIONAL}
                                                    -- depending on kdSchemeOID
-- Identifier is defined in 3.1.
```

initiatorKeyBlock
A block of information from which the initiator can obtain either a secret key or a private key. When it is a secret key, it is a basic key usable between the initiator and the AEF of the requested target X-Server application component. Additional information for caching purposes may also be present. When it is a private key, it is either a temporary private key (if OtherKeyBlock contains an ECMA Generalised User Certificate - ucECMA) or a long term private key (if OtherKeyBlock contains a Directory user certificate - ucDirectory).

otherKeyBlock
This has three possible forms. It may contain a temporary Generalised User Certificate issued by the KD-Server. It may contain a long term public key certificate issued by a Certification Authority (CA). It may contain a targetKeyBlock which can be used to establish the basic key of the associated initiator key block with the AEF of a target X-Server application component.

keyConstructionData
Information contained in the initiatorKeyBlock which allows the value of either a secret key or a private key to be recovered. The keyConstructionData has two elements: an identifier of the method used (constructionMethodId) and parameters (methodParameters) relating to the method.

keyUseData
This information identifies for the initiator the other target X-Server application components which are sharing the same AEF. When the otherKeyBlock is a targetKeyBlock and only contains a targetPart then this information nominates the other target X-Server application components for which the basic key being established is also valid. It is used for caching purposes to establish a basic key cache between the initiator and the target AEF.

validity
The validity period of the basic key is optionally put in the initiator key block to enable the initiator to perform a new GetKI operation when the basic key has expired.

constructionMethodId
An object identifier which identifies the protection method to be used for protecting the key transmitted within the InitiatorKeyBlock.

methodParameters
Parameters to be used with the construction method. Two methods are explicitly supported but the Standard is open to other methods.

The first method, **oneWay**, uses a one way hash function with a defined algorithm identifier, **hashAlgId**, which is specified with a **seed** and optionally a **keySize** and a **keyIdentifier**. The value of the key is computed from the seed and the long term source key identified by the identifier (or a default source key) using the hash function identified by hashAlgId. The result of the hash function is then truncated to fit the key size requirement.

NOTE
This method is intended for basic key establishment with a target AEF that does not use a reversible cryptographic algorithm for basic key establishment.

The second method, **encrypted**, uses an encryption algorithm with a defined algorithm identifier, **symmetricAlgId**, which is specified and used with a key identified by a **keyIdentifier**. When no key identifier is specified, the basic key which protects the operation is used as an encryption key.

ucECMA
This is a Generalised User Certificate which contains a temporary public key issued by the KD-Server. It may be used by the initiator to build its own targetKeyBlock structures. Its syntax is given in Clause 8. It is issued by the KD-Server instead of a CA.

ucDirectory
This is a user certificate which contains a long term public key issued by a CA. It may be used by the initiator to build its own targetKeyBlock structures (and also for other purposes). It takes the form of a [ISO/IEC 9594-8] user certificate.

targetKeyBlock
A block which contains information which is to be sent to the AEF of an X-Server application component to establish a basic key with it.

initiatorKDSname
Name of the KD-Server used by the initiator.

kdSchemeOID
Identifies the key distribution scheme used. Allows the target AEF 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 KD-Server of the target AEF. This part is sent by the target AEF to its local KD-Server, using the ProcessKI operation in order to get the key which is in it. It must always contain the name of the application. The mapping between the name of the application and the name of the target AEF is known to the target AEF's KD-Server 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 (see the ProcessKI operation described in part 3). targetKDSpart should include data that enables the KD-Server of the target AEF to authenticate the KD-Server of the initiator. When the "Primary Principal Qualification" protection method needs to be used for the PAC, unless there is an accompanying targetPart, targetKDSpart must contain 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 target KDSpart has been processed by the KD-Server 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 KD-Server 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.

10.3 Example walkthroughs of key exchanges

This clause gives two examples corresponding to configurations 1 and 6 described in 10.1. The handling of Primary Principal qualifier attribute values used in the certificate protection method known as Primary Principal Qualification (described in 7.2) is not described, though when present they would be carried within the targetPart constructs described.

10.3.1 Notation

A ---> B : X + Y	A sends X and Y to B
BK	Basic Key
DKB	Dialogue Key Block
IKB	Initiator Key Block
K	Long Term Secret Key
Pri[X]	Private Key of X
Pub[X]	Public Key of X
TKB	Target Key Block
(M)X	Data M encrypted with X
{M}X	Data M signed with X

The name of the X-Server Application Component controlled by the target AEF is App-X.

10.3.2 Example 1

This clause describes step by step the establishment of a basic key and dialogue keys between an initiator and a target AEF when the KD-Server used by the initiator shares a symmetric long term key with the target AEF.

The initiator and the KD-Server share a basic key (BK1), this basic key is assumed to have been established in a previous operation. To establish a basic key between the initiator and the target AEF, the steps are as follows :

1. The initiator requests Keying Information from the KD-Server for App-X with a GetKI operation. This operation is protected by BK1.

   Initiator -----> KD-Server : App-X name

2. The KD-Server generates a basic key for App-X 's target AEF (BK2) and returns it in the initiator key block encrypted with BK1, and in the target key block, encrypted with the long term secret key shared with the target AEF (K).

   KD-Server -----> Initiator : KeySet = IKB + TKB, where:
   IKB contains (BK2)BK1 in the field "encryptedKeyValue"
   TKB contains (BK2)K in the field "targetPart"

3. The initiator analyses the initiator key block, generates a dialogue key block and sends the target key block to the target AEF along with the dialogue key block. This operation is integrity protected by BK2. It also generates an integrity and a confidentiality dialogue key.

   Initiator -----> target AEF : DKB + TKB

4. The target AEF decrypts the target key block with K, and generates an integrity and a confidentiality dialogue key from BK2 and the dialogue key package, for App-X to use.

10.3.3 Example 2

This clause describes step by step the establishment of a basic key and dialogue keys between an initiator and a target AEF when the initiator and the target AEF have different KD-Servers, and the two KD-Servers possess private keys. The private keys are used to create a DESK which can subsequently be used as if the two KD-Servers shared a symmetric long term key,

The initiator and its KD-Server (KD-Server1) share a basic key (BK1), this basic key has been established in a previous operation. To establish a basic key between the initiator and App-X's target AEF, the steps are as follows :

1. The initiator requests Keying Information from KD-Server1 for App-X with a GetKI operation. This operation is protected by BK1.

   Initiator -----> KD-Server1 : App-X name

2. KD-Server1 generates a basic key for App-X's target AEF (BK2), and a DESK to be shared with App-X's KD-Server (KD-Server2). In the Initiator Key Block, KD-Server1 returns BK2 encrypted with BK1, and in the Target Key Block, BK2 encrypted with the DESK along with the DESK encrypted with the public key of KD-Server2 and App-X's name, signed by its own private key.

   KD-Server1 -----> Initiator : KeySet = IKB + TKB, where:
   IKB contains (BK2)BK1 in the field "encryptedKeyValue"
   TKB contains KD-Server1 name in the field "initiatorKDSname" and,
   (BK2)DESK + {(DESK)Pub[KD-Server2] + App-X name }Pri[KD-Server1]
   in the field "targetKDSpart"

3. The initiator analyses the initiator key block, generates a Dialogue Key Block and sends the target key block to the target AEF along with the Dialogue Key Block. This operation is integrity protected by BK2. It also generates an integrity and a confidentiality dialogue key for use later in communicating with App-X.

   Initiator -----> target AEF : DKB + TKB

4. The target AEF stores the dialogue key block and sends the "targetKdsPart" field of the Target Key Block, and KD-Server1's name, to KD-Server2 using the ProcessKI operation. This operation is protected by the long term key shared between KD-Server2 and the target AEF.

   target AEF -----> KD-Server2 :
   KD-Server1 + (BK2)DESK + {(DESK)Pub[KD-Server2]
   + App-X name} Pri[KD-Server1]

5. KD-Server2 checks that App-X is served by this target AEF, checks that the DESK and App-X name came from a KD-Server that it trusts, decrypts the DESK with its private key, decrypts BK2 with DESK and returns a target key block containing BK2 encrypted with the long term secret key shared with the App-X's target AEF (K).

   KD-Server2 -----> App-X's target AEF : (BK2)K

6. The target AEF decrypts BK2 using K, and generates an integrity and a confidentiality dialogue key from BK2 and the dialogue key package, for App-X to use.

If KD-Server1 and KD-Server2 cache the DESK, and reuse it for future access involving KD-Server1 and KD-Server2, there is no further need to use public key encryption for those accesses.

11 DialogueKeyBlock

DialogueKeyBlock 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. Dialogue keys are explained in part 1. The syntax is as follows:

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

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

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

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

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.

Annex A

(Normative)

Use of identities in the APA-Application

A.1 Uses of identity

Identity is a real world attribute which has multiple uses in a computer system. Here are six such uses:

– as a means of claiming who you are, i.e. it is what you present when logging on to a system to help the system locate the authentication information that it will use to verify your claim. Call this use "Logging on".

– as a means of making you accountable for your actions, i.e. it is the identity that appears in audit trails. Call this use "Audit".

– as a means of identifying who should pay for the use of the system. Call this use "Charging".

– as a means of obtaining access to protected objects, e.g. the identity which appears in an Access Control List. Call this use "Access".

– as a means of identifying you as the originator of a piece of data so that the receiver can prove to a third party that the piece of data really did originate from you. Call this "Non-repudiation".

– as a means of locating and permitting the release of your access Privilege Attributes within a security attribute application. This is a special case of the "Access" use. Call this use "Attribute Access".

All of these uses could be encompassed by implementing one single type of attribute: "Identity Attribute", used for everything; however if this were done there would be a number of security policies that would be impossible to implement, in fact all policies that required different values for different uses.

One simple example is where Tom is to act as if he were Jan for a period (Jan is on holiday), for only those things that Jan can do using only identity Jan as an identity attribute for "Access" purposes, i.e. as a Privilege Attribute in a PAC. He is permitted to do this with respect to accesses to all objects except access to Jan's other Privilege Attributes, so Tom cannot use Jan for "Attribute Access" purposes. Tom must still be accountable as himself for his actions when acting in this restricted way for Jan. In this example Tom could be used for "Logging on", "Audit", "Charging", "Non-repudiation" and "Attribute Access", but Jan would be used for "Access" use.

A.2 Identity attributes

The question now arises: In order to support a reasonable spectrum of real security policies, do we have to provide separate types of Identity Attribute for each use? The answer in principle is: for five of them, yes, but the identity used for logging on need not appear other than as a parameter in authentication exchanges. Most of the attributes should be optional. The names that have been chosen for the attribute types in this Standard are as follows:

– Authenticated Identity, which is used in an AUC for "Attribute Access",

– Audit Identity, which is used in both AUCs and PACs for "Audit",

– Charging Identifier, which is used in PACs for "Charging". The term "identifier" is used in preference to "identity" because the value will commonly be the identification of an account rather than of the individual himself,

– Access identity, which is used in PACs for "Access",

– Non-repudiation Identity, which is used in PACs for "Non-repudiation".

The name used when "Logging on" is known in this Standard as Logon Name. The term "name" is used to distinguish it from other forms of identity because unlike them it is not stored in certificates, it is simply a name used by a claimant as a convenient form of identification.

A proof for non-repudiation serves the purpose of solving a dispute between two human beings. It is thus important that the individual involved in the proof can be uniquely identified by a third party, normally another human being. The "Non-repudiation Identity" serves this purpose. To ensure the uniqueness of the Non-repudiation identity, a second system orientated component of the identity may be required. Its value might for example be that of the user's

Authenticated Identity, which will be unique within the domain of the A-Service that provided it. So one component may be the identity by which the individual is known and represented by the system, and another is that by which he is known by humans.

A.3 Rules for the inclusion and omission of identity attributes from AUCs and PACs

The following rules - subject to security model and policy considerations - are applied to determine which identity attribute types are put into AUCs and PACs and in what combinations:

1. None, one or more than one of each identity attribute of type "Access" or "Non-repudiation" is permitted in a PAC. They are not permitted in an AUC.
2. None, one or more identity attributes of type "Audit" is permitted in a PAC or an AUC. If more than one is present, all of them are intended to be included in audit trails.
3. None or at most one identity attribute of type "Charging" is permitted in a PAC or an AUC.
4. The "Authenticated Identity" attribute is optional at most once in a PAC and mandatorily appears exactly once in an AUC.
5. If Audit Identity is missing from either an AUC or a PAC then the use of other identity types, where present, for audit purposes is a matter of Audit Policy. Under some anonymity Audit Policies it is possible that for identification purposes only, the certificate identifier field appears in audit trails of subsequent accesses which use the AUC or PAC. This would then act as a link to the accessor's identity, via information which appears only in an APA-Application audit trail.
6. If access identity is not present in a PAC, other identities cannot act as access identities and must not be used in access control decisions.
7. If Non-repudiation Identity is not present, any of the other Identity types may be used for non-repudiation purposes, depending on security policy.

These rules permit simple policies to produce simple AUCs and PACs, typically with only one identity in them.

Annex B

(Normative)

Summary of ASN.1 (alphabetical order)

```
SecurityInformationObjects {  iso(1) identified-organisation(3) icd-ecma(0012)
                  standard(0) apa(219) modules(1) securityInformationObjects(3) }

DEFINITIONS ::=

BEGIN
-- exports everything

IMPORTS

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

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

-- End of imports

-- Security Information Objects in alphabetical order

AUCSpecificContents ::= SEQUENCE {
    aucSyntaxVersion      [0]   INTEGER    {   version1 (1)}              DEFAULT 1,
    protectionMethods     [1]   SEQUENCE OF MethodGroup                   OPTIONAL,
    aucType               [2]   ENUMERATED    {  primaryPrincipal             (1),
                                                 secondaryPrincipal           (2)},
    authenticationLevel   [3]   INTEGER                                   OPTIONAL,
    aucAtts               [4]   SEQUENCE OF SecurityAttribute,
    timePeriods           [5]   TimePeriods                               OPTIONAL}

CertandECV ::=     SEQUENCE {
    certificate              [0]    GeneralisedCertificate,
    ecv                      [1]    ECV,                                  OPTIONAL}
                  -- ECV is defined in clause 9
```

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

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

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

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

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

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

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

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

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

GUCSpecificContents ::= SEQUENCE{
                                             attributes     [0]      SEQUENCE OF SecurityAttribute,
                                             publicKey      [1]      BIT STRING}

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}}

InitiatorKeyBlock ::= SEQUENCE {
         keyConstructionData   [0]      KeyConstructionData,
         keyUseData            [1]      KeyUseData                    OPTIONAL,
         validity              [2]      Validity                      OPTIONAL}
                                                        -- defined in 3.1

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

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

KeySet ::= SEQUENCE {
         initiatorKeyBlock     [0]      InitiatorKeyBlock,
         otherKeyBlock         [1]      OtherKeyBlock}

KeyUseData ::= SEQUENCE OF TargetName

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),
                                              delegateQualification               (5),
                                              tracePointer                        (6),
                                              initiatorQualification              (7),
                                              count                               (8),
                                              checkBack                           (9),
                                              nestedChain                         (10)},
      otherMethod        [1] OBJECT IDENTIFIER}

MethodParameters ::= CHOICE {
         oneWay                [0]     OneWayParameters,
         encrypted             [1]     SEQUENCE {
                  encryptedKeyValue      [0]     BIT STRING,
                  keyIdentifier          [1]     INTEGER                 OPTIONAL,
                  symmetricAlgId         [2]     AlgorithmIdentifier     OPTIONAL},
         other                 [2]     ANY}             -- defined by constructionMethodId

Mparm ::= CHOICE{
      pValue                   [0]     PValue,
      securityAttribute        [1]     SecurityAttribute,
      integerValue             [2]     INTEGER,
      octetValue               [3]     OCTET STRING,
      bitStringValue           [4]     BIT STRING,
      nestedCertificate        [5]     CertandECV,
      printableStringValue     [6]     PrintableString,
      certificateId            [7]     CertificateId,-- defined in 3.4
      otherValue               [8]     SEQUENCE{
                                          paramType  [0] OBJECT IDENTIFIER,
                                          paramValue [1] ANY -- defined by paramType}

OneWayParameters ::= SEQUENCE {
         seed                  [0]     BIT STRING,
         keySize               [1]     INTEGER                           OPTIONAL,
         hashAlgId             [2]     AlgorithmIdentifier               OPTIONAL,
         sourceKeyId           [3]     INTEGER                           OPTIONAL }

OtherKeyBlock ::= CHOICE {
         ucECMA                [0] GeneralisedCertificate,
         ucDirectory           [1] Certificate,       -- imported from the Directory Standard
         targetKeyBlock        [2] TargetKeyBlock}
```

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

PrivilegeAttribute ::= SecurityAttribute

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

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 in this Standard.
        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

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

```
SecurityAttribute ::= SEQUENCE{
        attributeType    Identifier,
        attributeValue   SET OF SEQUENCE{
                                   definingAuthority   [0] Identifier              OPTIONAL,
                                   securityValue       [1] SecurityValue} }
        --  NOTE: SecurityAttribute is not tagged, for compatibility with the Directory
                  Standard.

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

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

Signature ::= SEQUENCE{
     signatureValue         [0]    BIT STRING,
     asymmetricAlgId            [1]    AlgorithmIdentifier                    OPTIONAL,
     hashAlgId                  [2]    AlgorithmIdentifier                    OPTIONAL,
     issuerCAName               [3]    Identifier                             OPTIONAL,
     caCertInformation          [4]    CHOICE {
                  caCertSerialNumber      [0]    INTEGER,
                  certificationPath       [1]    CertificationPath}           OPTIONAL}
--   AlgorithmIdentifier    is imported from [ISO/IEC 9594-8]
--   CertificationPath      is imported from [ISO/IEC 9594-8]

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

```
TargetKeyBlock ::= SEQUENCE {
        initiatorKDSname      [0]    Identifier                    OPTIONAL,
        kdSchemeOID           [2]    OBJECT IDENTIFIER,
        targetKDSpart         [3]    ANY                           OPTIONAL,
                                               -- depending on kdSchemeOID
        targetPart            [4]    ANY                           OPTIONAL}
                                               -- depending on kdSchemeOID
-- Identifier is defined in 3.1.

TargetName ::= Identifier                 -- Identifier is defined in 3.1

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

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

END -- of SecurityInformationObjects
```

Annex C

(Informative)

Changes from the first edition (December 1994)

This second edition of Part 2 of Standard ECMA-219 has been changed from the first edition (December 1994) in the following respects:

1. The field: targetKDSname has been removed from the TargetKeyBlock syntax (see clause 10.2.1). Experience of defining specific key distribution schemes has shown that this value is better placed within targetKDSpart, details vaying depending on the specific scheme being defined.
2. A minor editorial error in clause 10.3 has been fixed. The term PPID was wrongly used instead of Primary Principal Qualifier Attributes.

Part 3 - Service definitions

1 Introduction

This part provides the Service Definitions of the APA-Application. It contains the abstract model, common arguments, common operations and definitions of the Authentication Port and the Privilege Attribute Port.

Except for the specification of a management port, the three parts together of this Standard provide a complete description of the external interfaces of the APA-Application. Relaying or referral to other servers of the same APA-Application is supported through these interfaces, but the definition of the internal interfaces of such distributed implementations is out of scope of this Standard.

2 APA abstract model

2.1 The APA-Application

The APA-Application can be described without reference to its internal structure. The services provided are made available by the APA-Application object at its ports. A type of port represents a particular view of the APA-Application object.

In this abstract model the application user is also modelled as an object, the APA-Client object. It obtains the services provided by the APA-Application through the ports which are paired with APA-Application ports of the same type. The functional model which underlies these services is given in part 1.

The APA-Client object accesses the APA-Application through access points containing one, two or three ports. Each port corresponds to a set of abstract operations. All ports are asymmetric; the APA-Application object takes the role of supplier and the APA-Client object the role of consumer. The APA-User is either a Primary or Secondary Principal. The type of principal acting as the APA-User is transparent to the APA-Client. Figure 1 illustrates this.

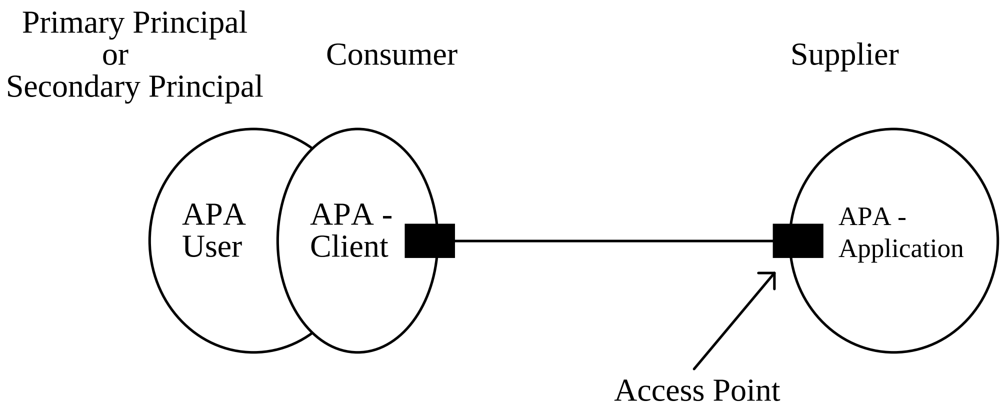


Figure 1. APA Abstract Model

The APA-Application object supports three types of port:

- the authentication port (A-port),
- the privilege attribute port (PA-port),
- the key distribution port (KD-port).

The APA-Application model is shown in figure 2, though not all permutations of port usage are illustrated.

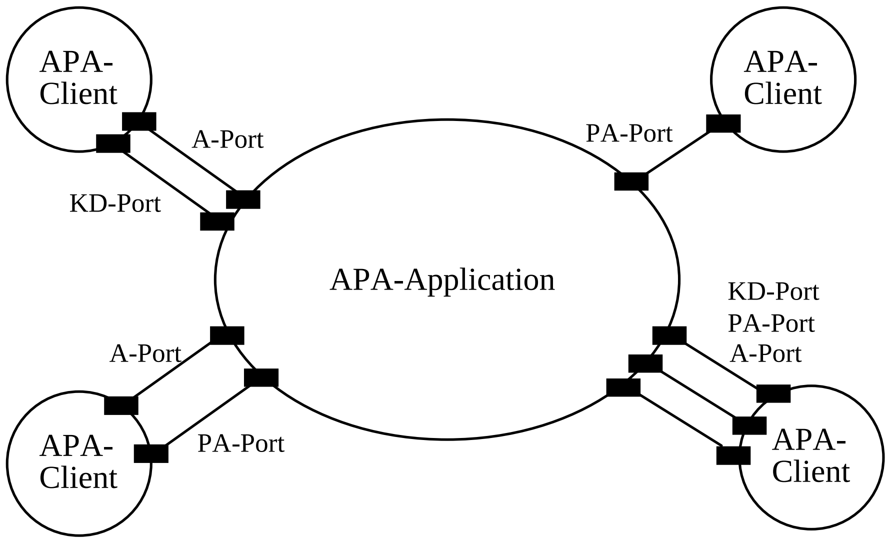


Figure 2. APA Application Model

The APA-Client object may access one, two or possibly three ports at the same access point depending on configuration. See part 1 for a description of different configurations. The APA-Client object may thus use the A-port and PA-Port in three ways. It can:

1. use both ports through the same access point on a collocated A- and PA-Server.
2. use a single port though an access point on a collocated A- and PA-Server.
3. use one port at the access point of a separate A- or PA-Server providing one port only.

Figure 3 illustrates this. All the variations shown can use an additional KD-Port on the APA-Application

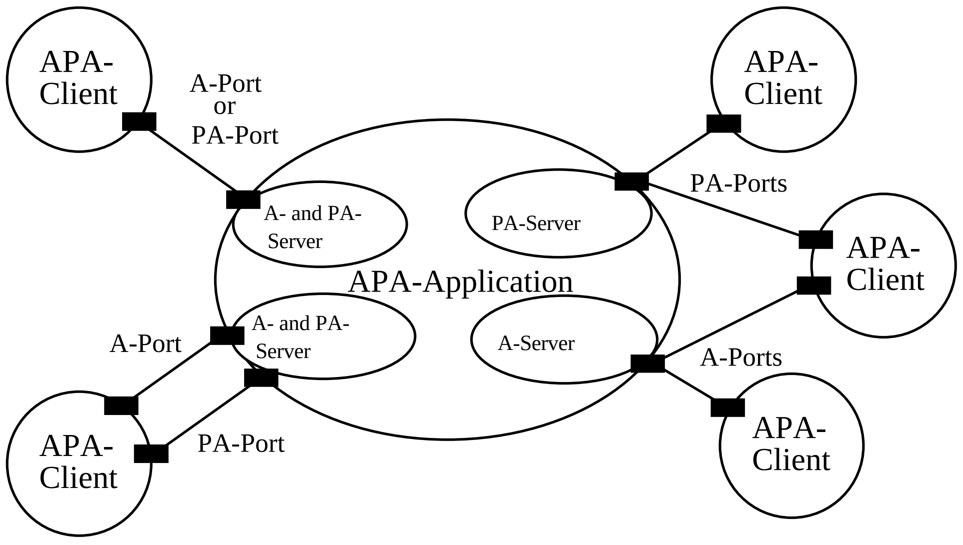


Figure 3. APA-Application Model with servers

The APA-Application is defined by the following:

```
APA-Application OBJECT
    PORTS {  authentication-port        [S],
             privilege-attribute-port   [S],
             key-distribution-port      [S]}
    ::= id-ot-apa-application
```

```
APA-Client OBJECT
    PORTS {  authentication-port        [C],
             privilege-attribute-port   [C],
             key-distribution-port      [C]}
    ::= id-ot-apa-client
```

2.1.1 APA ports

The abstract ports of the APA-Application are defined below. Together these ports provide the full functionality of the APA-Application. Some operations appear only in combination with others. These are given below. Short names used for the operations when in combination are given in brackets where relevant:

- DeclareOperationContext (Declare),
- Operations which require the authorisation and/or protection provided by DeclareOperationContext or by OpenSA. These are: GetKI, ChangePassword (ChangePW), ContinueChangePassword (ContChangePW), GetASName, ConfirmPresence (Confirm), RevokeCertificate (Revoke), and ProcessKI.
- CloseSA (Close).

Thus "DeclareandClose", for example, indicates a combined operation containing both DeclareOperationContext and CloseSA. Combined operations are further described in clause 8.

Operations are defined in terms of basic components ("atomic" operations) which optionally can be combined together to make "combined" operations. In the abstract model, these operations are provided by the Authentication port, the Privilege Attribute port and the Key Distribution Port. All of these ports can support both atomic and combined operations. The precise combination rules are specified separately for each port in clause 8.

2.1.1.1 APA authentication port

The Authentication Ports support operations for the authentication of APA-clients and the subsequent handling of Authentication Certificates. Other operations enable clients to change their passwords, and the APA-Application to enquire whether an authenticated principal is still present. Common operations are also supported to obtain Keying Information used for establishing Security Associations, and to operate over such associations. These operations are defined in detail in clauses 4. and 5.

```
authentication-port PORT
CONSUMER INVOKES {
        OpenSA,
        Authenticate,
        ContinueAuthentication,
        CheckAUC,
        DeclareandAuth,
        DeclareandContAuth,
        DeclareandChangePW,
        DeclareandContChangePW,
        DeclareandCheckAUC,
        DeclareandGetASName,
        DeclareandGetAT,
        DeclareandGetATandGetKI,
        DeclareandGetKI,
        DeclareandOpen,
        DeclareandClose,
        DeclareandProcessKI,
        OpenandAuth,
        OpenandChangePW,
        OpenandCheckAUC,
        OpenandGetASName,
        OpenandGetAT,
        OpenandGetATandGetKI,
        OpenandGetKI,
        OpenandProcessKI}

SUPPLIER INVOKES {
        DeclareandConfirm,
        DeclareandRevoke,
        DeclareandClose}
        ::= id-pt-a
```

2.1.1.2 APA Privilege Attribute port

The Privilege Attribute Port provides a number of operations for issuing Privilege Attributes and Keying Information to clients that posses the appropriate proof of authentication.

```
privilege-attribute-port PORT
CONSUMER INVOKES {
    OpenSA,
    CheckPAC,
    RefinePAC,
    DeclareandGetACT,
    DeclareandGetACTandGetKI,
    DeclareandGetKI,
    DeclareandCheckPAC,
    DeclareandProcessKI,
    DeclareandRefinePAC,
    DeclareandOpen,
    DeclareandClose,
    OpenandGetACT,
    OpenandGetACTandGetKI,
    OpenandGetKI,
    OpenandCheckPAC,
    OpenandProcessKI,
    OpenandRefinePAC}

SUPPLIER INVOKES {
    DeclareandClose}
    ::= id-pt-pa
```

2.1.1.3 APA Key Distribution port

The Key Distribution Port provides a number of operations for distribution of Keying Information to initiators and targets AEFs that posses the appropriate SA or basic key.

```
key-distribution-port PORT
CONSUMER INVOKES {
    OpenSA,
    DeclareandGetKI,
    DeclareandProcessKI,
    OpenandGetKI,
    OpenandProcessKI,
    DeclareandOpen,
    DeclareandClose}

SUPPLIER INVOKES {
    DeclareandClose}
    ::= id-pt-kd
```

3 Specification of bind and unbind operations

This clause defines the A-bind, A-unbind, PA-bind, PA-unbind, KD-bind and KD-unbind operations used to establish and release abstract associations between the APA-Client and the APA-Application.

3.1 A-bind

This abstract operation enables an APA-Client to establish an abstract communications association with the APA-Application, or the APA-Application to establish an abstract communications association with an APA-Client. The A-bind can only bind client and application A-ports together. The A-bind indicates the application context of the association (A-association). An A-association can only be released, using A-unbind, by the initiator of that association. Abstract operations other than A-bind can only be invoked in the context of an established A-association. The successful completion of the A-bind signifies the establishment of an A-association. The disruption of an A-bind by a bind-error indicates that the A-association has not been established.

The A-association can be established without an existing Security Association between the initiator and responder. The release of an A-association has no implications for the existence of Security Associations.

The A-bind operation is specified by

```
A-bind ::= ABSTRACT-BIND
TO { authentication-port }
    BIND
    ARGUMENT        AbindArgument
    RESULT          AbindResult
    BIND-ERROR      AbindError

AbindArgument ::= SEQUENCE {
    protocolVersionID       ProtocolVersionID,
    serviceType             ServiceType                         OPTIONAL}

ProtocolVersionID   ::= INTEGER

ServiceType         ::= SEQUENCE {
    referralSupported       [0]   BOOLEAN,
    stateSupported          [1]   ENUMERATED {  stateless       (1),
                                                stateful        (2),
                                                both            (3)}
    profileSupported        [2]   OBJECT IDENTIFIER             OPTIONAL}

AbindResult ::= SEQUENCE {
    clientInformation       [0]   STRING                        OPTIONAL,
    serviceType             [1]   ServiceType}

AbindError ::= ENUMERATED {
    busy                          (1),
    serviceTypeUnsupported        (2),
    unspecified                   (3)}
```

3.1.1 Arguments

The A-bind has two arguments, the protocol version identifier and the service type.

protocolVersionID
The protocol version identifier contains the version of the APA Application protocol the initiator of the association wants to use.

serviceType
The service type in the argument indicates what modes of distributed operation of APA-operations are supported by the initiator:

referralSupported defines whether the initiator supports referral or not. Chaining and referral are described in part 1.

stateSupported defines what state property of A-Servers are supported by the initiator. An initiator can either support only a stateless A-Server, or only a stateful one, or both.

profileSupported defines what implementation profile of this Standard is supported by the initiator.

3.1.2 Results

The results of the A-bind can return free format information to the initiator, such as news, and information concerning the service type to be provided, depending on security policy.

clientInformation
The client information field contains free format information such as news, to be delivered to the initiator when it is an A-Client acting on behalf of a Primary or Secondary Principal.

serviceType
The service type in the result indicates what mode of distributed operations are supported by the responder. The value of this argument is independent of any value of service type specified in the A-Bind argument.

3.1.3 Errors

The A-bind may be disrupted by the following errors.

The busy bind-error indicates the failure of association establishment because the responder is busy.

The serviceTypeUnsupported bind error indicates that the choice(s) made by the caller is(are) not supported.

An unspecified bind-error is used when the responder is unwilling or unable to indicate the reason for the error.

3.2 A-unbind

This abstract operation enables the release of an established A-association by the initiator of the association. If the target service is a connectionless one, unbind just has a local effect.

```
A-unbind ::= ABSTRACT-UNBIND
FROM { authentication-port }
```

This operation has no arguments.

3.3 PA-bind

This abstract operation enables an APA-Client to establish an abstract communications association with the APA-Application, or the APA-Application to establish an abstract communications association with an APA-Client. The PA-bind can only bind client and application PA-ports together. The PA-bind indicates the application context of the association (PA-association). A PA-association can only be released, using PA-unbind, by the initiator of that association. Abstract operations other than PA-bind can only be invoked in the context of an established PA-association. The successful completion of the PA-bind signifies the establishment of a PA-association. The disruption of a PA-bind by a bind-error indicates that the PA-association has not been established.

The PA-association can be established without an existing Security Association between the initiator and responder. The release of a PA-association has no implications for the existence of Security Associations.

The PA-bind operation is specified by

```
PA-bind ::= ABSTRACT-BIND
TO { privilege-attribute-port }
    BIND
    ARGUMENT    PAbindArgument
    RESULT      PAbindResult
    BIND-ERROR  PAbindError
```

```
PAbindArgument ::= AbindArgument

PAbindResult ::= AbindResult

PAbindError ::= AbindError
```

3.3.1 Arguments

The use of the PAbindArgument is analogous to the use of the AbindArgument on the A-port. In a PAbindArgument, the field referralSupported in the serviceType argument is always set to FALSE.

3.3.2 Results

The use of the PA-bind result is the same as for the A-bind, except that in a PAbindResult, the field referralSupported in the serviceType argument is always set to FALSE.

3.3.3 Errors

The use of PAbindError is the same as AbindError, defined in 3.1.3.

3.4 PA-unbind

This abstract operation enables the release of an established communication association by the initiator of the association. If the target service is a connectionless one, unbind just has a local effect.

```
PA-unbind ::= ABSTRACT-UNBIND
FROM { privilege-attribute-port }
```

This operation has no arguments.

3.5 KD-bind

This abstract operation enables an APA-Client to establish an abstract communications association with the APA-Application, or the APA-Application to establish an abstract communications association with an APA-Client. The KD-bind can only bind client and application KD-ports together. The KD-bind indicates the application context of the association (KD-association). A KD-association can only be released, using KD-unbind, by the initiator of that association. Abstract operations other than KD-bind can only be invoked in the context of an established KD-association. The successful completion of the KD-bind signifies the establishment of a KD-association. The disruption of a KD-bind by a bind-error indicates that the KD-association has not been established.

The KD-association can be established without an existing Security Association between the APA-Client and the APA-Application. The release of a KD-association has no implications for the existence of Security Associations.

```
KD-bind ::= ABSTRACT-BIND
TO { key-distribution-port }
    BIND
    ARGUMENT    KDbindArgument
    RESULT      KDbindResult
    BIND-ERROR  KDbindError

KDbindArgument ::= SEQUENCE {
    protocolVersionID       ProtocolVersionID,
    kdServiceType           KDserviceType                       OPTIONAL}

KDserviceType      ::= SEQUENCE {
    stateSupported       [1]    ENUMERATED {  stateless        (1),
                                                 stateful      (2),
                                                 both          (3)}
    profilesSupported    [2]    OBJECT IDENTIFIER               OPTIONAL}

KDbindResult ::= ServiceType
```

```
KDbindError ::= AbindError
```

3.5.1 Arguments

The use of the KDbindArgument is analogous to the use of the AbindArgument on the A-port. In a KDbindArgument, the field referralSupported in the serviceType argument is not present. The other arguments present are used as specified in clause 3 for the A-port.

3.5.2 Results

The result of the KD-bind is the service type supported by the APA-application. The meaning of the arguments that are present is the same as that given for the A-bind result in 3.1.2.

3.5.3 Errors

The use of KDbindError is the same as AbindError, defined in 3.1.3.

3.6 KD-unbind

This abstract operation enables the release of an established communications association by the APA-client. If the APA-Service is a connectionless one, unbind just has a local effect.

```
KD-unbind ::= ABSTRACT-UNBIND
FROM { key-distribution-port }
```

This operation has no arguments.

4 Common atomic operations

This clause defines operations supported by all of the ports of the APA Application.

4.1 Open Security Association (OpenSA)

This abstract operation supports the initiation of an SA on a port of the APA-Application.

```
OpenSA ::= ABSTRACT-OPERATION
    ARGUMENT   OpenSAArgument
    RESULT     OpenSAResult
    ERRORS     {  AssocAlreadyOpen
                  AUCSpecificError,
                  CertificateError,
                  InsufficientAuthorisation,
                  InvalidDialogueKeyBlock,
                  InvalidKeyBlock,
                  OperationNotSupported,
                  PACSpecificError,
                  TimingFailure,
                  Unspecified}

OpenSAArgument ::= SEQUENCE{
    SAIdentifier      [0]   OCTET STRING,
    targetKeyBlock    [1]   TargetKeyBlock,                      -- defined in part 2
    dialogueKeyBlock  [2]   DialogueKeyBlock    OPTIONAL,        -- defined in part 2
    authorisation     [3]   Authorisation,              -- to be associated with the SA
    protection        [4]   Protection          OPTIONAL
```

```
Authorisation      ::= SEQUENCE {
    callingPrincipalInfo        [0]     PrincipalInfo         OPTIONAL,
    ppInfo                      [1]     PrincipalInfo         OPTIONAL}

PrincipalInfo      ::= CHOICE {
    aucInfo                     [0]     CertandECV,
    pacInfo                     [1]     CertandECV}
                   -- CertandECV is defined in part 2

Protection         ::= CHOICE {
        timestamp               [0]     UniqueNumber,
        saSeqNumber             [1]     INTEGER VALUE 1}

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

OpenSAResult   ::= SEQUENCE {
        SAIdentifier            [0]     OCTET STRING,
        userInformation         [1]     Printable String      OPTIONAL,
        protection              [2]     Protection            OPTIONAL}
```

4.1.1 Arguments

SAIdentifier
A random number for constructing an identifier for the Security Association being formed; it is one which (with high probability) has not been used previously. This random number is generated by the client. The server generates its own random number and concatenates it to the end of the client's random number. The concatenated value is then taken to be the identifier of the Security Association being established.

targetKeyBlock
A targetKeyBlock obtained in an otherKeyBlock from a previous call on the APA-Application or derived from information contained in it.

dialogueKeyBlock
Optional information used in conjunction with the basic key to establish a confidentiality dialogue key. The use of dialogue keys is explained in part 1.

authorisation
Authorisation information to be associated with the SA, and which will be used to authorise future operations issued over this SA.

callingPrincipalInfo
This argument is used to carry the AUC or the PAC of the principal issuing the operation. This may be either a Primary Principal or a Secondary Principal.

The construct used for this is CertandECV which identifies the authorising certificate and the External Control Value(s) (ECV) for the certificate which must be present in order to authorise its use. The different possible types of External Control Values depend on the protection methods being applied. These are described in part 2.

ppInfo
This argument can be used when a Secondary Principal is issuing the operation. It carries the Primary Principal AUC or the Primary Principal PAC and enables attributes associated with the Primary Principal through which the Secondary Principal is operating to be taken into account. The AUC can be presented only if the port in use supports the provision of PACs for the Primary Principal. This saves the cost of obtaining the Primary Principal's PAC in advance, simply in order to present it back to the server. If this argument is omitted, and information relating to the Primary Principal is not already available to the APA-Application for

this SA, the latter assumes that the Primary Principal through which the Secondary Principal is asking for the PAC is unknown, and may impose appropriate constraints on the contents of any PACs returned according to security policy. When a Primary Principal does not want its full set of attributes to be taken into account, it should not use its AUC, but should first obtain a PAC with the required attributes in it.

protection
Indicates the form of replay protection to be applied to this operation and subsequent operations issued in the context of this SA, as identified below:

timestamp
Contains a unique number which consists of the date and time at which the operation was created concatenated with a random number. If the operation is time stamped, the response must also be.

saSeqNumber
The sequence number of the message within the SA, starting at the value 1. Defends against replay and message deletion attacks. If operations contain sequence numbers, the responses must do so also. The response to a numbered message contains the same number.

4.1.2 Results

The return of a result indicates the successful establishment of an SA with the APA-Application. The result carries the **SAIdentifier** after the server's value has been added (see the description above of SAIdentifier in OpenSAArgument, and optionally **userInformation**, which is information intended to be displayed as received to authenticating Secondary Principals who are human users, such as news about the operation of the server with which the SA is to be established. The contents of the **protection** field reflect the protection requirements expressed in the operation argument (q.v.).

4.1.3 Errors

The errors that may disrupt the Open SA abstract operation are listed below, and for each abstract error the clause in which it is defined is listed. Authorisation failures due to the inability to establish any access rights are recorded as errors against this operation rather than against each operation with which it may be combined. However if authorisation was successfully established and an operation using this authorisation then failed on an access control check, this would be attributed to the operation in question.

Abstract error	Clause
AssocAlreadyOpen	4.7.1
AUCSpecificError	4.7.2
CertificateError	4.7.3
InsufficientAuthorisation	4.7.8
InvalidDialogueKeyBlock	4.7.11
InvalidKeyBlock	4.7.12
OperationNotSupported	4.7.15
PACSpecificError	4.7.16
TimingFailure	4.7.19
Unspecified	4.7.21

4.2 DeclareOperationContext

This abstract operation is only used in combination with other atomic operations (since otherwise there is nothing for which to declare the context). It always appears first. For stateful services, it identifies the SA within which the combined operation of which it is part is being issued, along with other Security Association information such as a timestamp or the message sequence number within the SA. For stateless services, it provides Keying Information to protect the combined operation of which it is part, along with protection information such as a timestamp.

```
DeclareOperationContext ::= ABSTRACT-OPERATION
    ARGUMENT        DeclareOperationContextArgument
    RESULT          DeclareOperationContextResult
    ERRORS          {   AUCSpecificError,
                        InvalidDialogueKeyBlock,
                        InvalidKeyBlock,
                        OperationNotSupported,
                        PACSpecificError,
                        SAFailure,
                        TimingFailure,
                        Unspecified }

DeclareOperationContextArgument ::= SEQUENCE{
        operationContext    [0]   OperationContext,
        dialogueKeyBlock    [1]   DialogueKeyBlock           OPTIONAL,
        authorisation       [2]   Authorisation              OPTIONAL,
                                        -- Authorisation is defined in 4.1
        protection          [3]   Protection                 OPTIONAL}
                                        -- Protection is defined in 4.1

OperationContext ::= CHOICE {
        saId                [0]   INTEGER,          -- SA ID for stateful operation
        targetKeyBlock      [1]   TargetKeyBlock }  -- keying info for stateless operation

DeclareOperationContextResult ::= Protection                 OPTIONAL
```

4.2.1 Arguments

operationContext

saId
For stateful operation, an identifier of the SA being declared as the context.

targetKeyBlock
A targetKeyBlock obtained in an otherKeyBlock from a previous call on the APA-Application or derived from information contained in it. It is used when the APA-Server is stateless.

dialogueKeyBlock
Optional information used in conjunction with the basic key to establish a confidentiality dialogue key. The use of dialogue keys is explained in part 1.

authorisation
Authorisation information to be associated with the atomic operation(s) with which this DeclareOperationContext operation is combined. The individual components of this field are explained in 4.1.

DeclareOperationContext is always issued as part of a combined operation. The authorisation presented here replaces, for this combined operation only, the authorisation of any Security Association within which it is issued.

protection
If the DeclareOperationContext operation declares an existing SA, then this field contains the type of protection information established for the SA. In the absence of an existing SA, this field specifies the protection to be applied to this combined operation and its response, i.e. either a timestamp or an operation identifier.

4.2.2 Results

protection
contains a value appropriate to the protection established either for this operation or for the SA which the operation is declaring.

4.2.3 Errors

The errors that may disrupt the DeclareOperationContext operation are listed below, and for each abstract error the clause in which it is defined is listed. Authorisation failures due to the inability to establish any access rights are recorded as errors against this operation rather than against each operation with which it may be combined. However if authorisation was successfully established and an operation using this authorisation then failed on an access control check, this would be attributed to the operation in question.

Abstract error	Clause
AUCSpecificError	4.7.2
InvalidDialogueKeyBlock	4.7.11
InvalidKeyBlock	4.7.12
OperationNotSupported	4.7.15
PACSpecificError	4.7.16
SAFailure	4.7.17
TimingFailure	4.7.19
Unspecified	4.7.21

4.3 Get Keying Information (GetKI)

This abstract operation enables clients to get or to build basic keys whatever initial keys are used for key distribution by the APA-Servers or later targets. The main data structure used for key distribution is the Key Set defined in part 2.

```
GetKI ::= ABSTRACT-OPERATION
    ARGUMENT        GetKIArgument
    RESULT          GetKIResult
    ERRORS             {  ConstructTypesNotSupported,
                          InitiatorUnknown,
                          KITypeNotSupported,
                          OperationNotSupported,
                          TargetUnknown,
                          TrustInformationFailure,
                          Unspecified }

GetKIArgument ::= SEQUENCE {
    kiTypeRequired           [0]   ENUMERATED {  targetKeyBlock         (1),
                                                 ucECMA                 (2),
                                                 ucDirectory            (3),
                                                 trustInfo              (4)}
    initiatorName            [1]   Identifier                      OPTIONAL,
    targetName               [2]   Identifier                      OPTIONAL,
    keyConstructSupported    [3]   SEQUENCE OF KeyConstructType    OPTIONAL
    helpfulInformation       [4]   SEQUENCE OF Certificate         OPTIONAL

                                   -- Certificate is imported from [ISO/IEC 9594-8: 1990] }

KeyConstructType::= CHOICE {
    predefinedType           [0]   ENUMERATED {          oneWay        (1),
                                                         encrypted     (2),
                                                         keyIndex      (3) }
    otherType                [1]   OBJECT IDENTIFIER}
```

```
GetKIResult::= CHOICE {
      keySet                    [0]    KeySet,            -- defined in part 2
      trustInformation          [1]    TrustedAuthorities}

TrustedAuthorities   ::= SEQUENCE{
      root CA                   [0]    RootCA,
      branchCA                  [1]    CADistinguishedName,
      leafCA                    [2]    CADistinguishedName}

RootCA               ::= SEQUENCE{
      caName                    [0]    CADistinguishedName,
      publicKey Value           [1]    BIT STRING,
      validityperiod            [2]    UTCTime}
```

4.3.1 Arguments

kiTypeRequired

Nominates the type of Keying Information required. This can be either:

targetKeyBlock	or
ucECMA	or
ucDirectory	

the nomination of a key set containing a specific type of Other Key Block which the client can request according to the key technology it supports. The types of the Other Key Block determine the type of the Initiator Key Block returned. The types identified above correspond to definitions in part 2.

trustInfo
trust information relating to the use of a user certificate (see results below)

initiatorName
Nominates the name of the initiator of the GetKI operation, when the initiator and the KD-Server cannot share a basic key but instead share a long term secret key. The long term secret key is then used as a basic key between the initiator and the KD-Server. This happens in case of delegation when the targetis not itself an authenticated principal, but acts as a delegate to request Keying Information for another target AEF.

targetName
If the key set is to contain a Target Key Block, the target name for which the key set is requested must be specified here.

keyConstructSupported
Specifies the key construction method supported by the initiator to recover the key from an initiatorKeyBlock of the keySet construct.

helpfulInformation
Information on public key values that may be available to the caller and useful to the KD-Server.

4.3.2 Results

keySet
When the kiTypeRequired in the GetKI request specifies a type of Other Key Block, this contains the requested Keying Information in the form of a key set as described in part 2.

trustInformation
When the kiTypeRequired in the GetKI request specifies trust information, this information informs the caller of the trust conditions under which a user certificate should be verified. The AS may distinguish between:

- trust "points" where both the name of the CA and the value of the public key of the CA are known,
- trust "branches" where CAs are trusted both to issue certificates relative to other authorities and to issue certificates to the users under their administrative authority,

– trust "leaves" where CAs only trusted to issue certificates to the users under their administrative authority.

4.3.3 Errors

The errors that may disrupt the GetKI operation are listed below, and for each abstract error the clause in which it is defined is listed.

Abstract error	Clause
ConstructTypesNotSupported	4.7.5
InitiatorUnknown	4.7.7
KiTypeNotSupported	4.7.14
TargetUnknown	4.7.18
TrustInformationFailure	4.7.20
Unspecified	4.7.21

4.4 Process Keying Information (ProcessKI)

This abstract operation enables a target AEF to get information needed to establish a basic key with an X-Client using the Target Key Block sent by the X-Client. The operation is only used when the X-Client and the target AEF are using different KD-Servers, and when the targetKDSpart field is present in the received Target Key Block.

```
ProcessKI ::= ABSTRACT-OPERATION
    ARGUMENT      ProcessKIArgument
    RESULT        ProcessKIResult
    ERRORS          {  InitiatorKDSUnknown,
                       KdSchemeNotSupported,
                       OperationNotSupported,
                       TargetUnknown,
                       Unspecified }

ProcessKIArgument ::= SEQUENCE {
    kdSchemeOID              [0]    OBJECT IDENTIFIER,
    initiatorKDSname         [1]    Identifier,
    targetKDSpart            [2]    ANY,      -- Defined by kdSchemeOID. See part 2
    targetAEFName            [3]    Identifier}

ProcessKIResult::= SEQUENCE {
    processedKeyBlock        [0]    ANY}      -- Defined by kdSchemeOID. See part 2
    InitiatorKDSdomain       [1]    Identifier}
```

4.4.1 Arguments

kdSchemeOID
Identifies the key distribution scheme used. Corresponds to definitions in part 2.

initiatorKDSname
Name of the KD-Server used by the X-Client

targetKDSpart
Part of the Target Key Block that needs to be passed to the calling target AEF's KD-Server.

targetAEFName
Name of the target AEF for which the Keying Information should be translated.

4.4.2 Results

processedKeyBlock
Contains Keying Information translated by the KD-Server into Keying Information for direct use by the target AEF for basic key establishment.

initiatorKDSdomain
Enables the target AEF to obtain the name of the domain of the initiator's KDS, so that the target AEF can compare the value against the issuer-domain name in the PAC, and ensure that it is equal. This prevents an attack by which a rogue but known external domain can steal PACs from another external domain.

4.4.3 Errors

The errors that may disrupt the ProcessKI operation are listed below, and for each abstract error the clause in which it is defined is listed.

Abstract error	Clause
InitiatorKDSUnknown	4.7.6
KdSchemeNotSupported	4.7.13
OperationNotSupported	4.7.15
TargetUnknown	4.7.18
Unspecified	4.7.21

4.5 Close Security Association (CloseSA)

This abstract operation supports the termination of an SA which may have been opened as a result of successful authentication to an A-Server or by means of an explicit OpenSA operation to a PA- or KD-Server. The implication of this operation is that the principal concerned has logged off and the state information held by the caller relating to the SA may have been destroyed. Neither party is expected to keep any state information relating to this SA. The SA that is closed is the one identified in the DeclareOperationContext operation with which this operation is always combined.

```
CloseSA :: = ABSTRACT-OPERATION
    ARGUMENT    CloseSAArgument
    RESULT
    ERRORS      {   OperationNotSupported,
                    Unspecified}

CloseSAArgument ::=  Reason

Reason      ENUMERATED {logoff            (1),
                        done              (2),
                        timeout           (3),
                        notPresent        (4),
                        revocation        (5),
                        recovery          (6)}
```

4.5.1 Arguments

reason

The use of the reason argument depends on who the initiator is and which port the operation is used at. The possibilities are tabulated below.

Initiator	Port	logoff	done	time-out	notPresent	revocation	recovery
Client	A	X			X	X	X
Applic	A	X		X	X	X	X
Client	PA/KD		X			X	X
Applic	PA/KD			X		X	X

The reason for the initiator closing the SA has different meaning depending on the use.

logoff: When the A-Client closes an SA with the logoff reason it means that the SA identified in the argument is terminated, and the principal that used it has logged off from the A-Service in accordance with normal procedures.

When an A-Server closes an SA with the logoff reason, it means that the principal using it is forced to log off and will no longer be able to use the SA on the A-Port. The APA-Client shall no longer allow use of the SA identified in the argument, or any of those derived from it.

done: When a PA- or KD-Client closes an SA with the done reason it means that the SA will no longer be required for operations on this PA- or KD-Port.

timeout: The timeout reason is used by the APA-Application to notify a client that the SA is no longer valid due to a timeout (derived SAs are also no longer valid).

notPresent: This reason for closing an SA is used by an APA-Client to notify the APA-Application that the principal related to the SA is no longer considered to be present at the client, so the SA can be closed in a normal way.

revocation: When an A-Client or PA-Client closes an SA with this reason it is because a (management) operation on the client has revoked it to invalidated it's use and use of all SAs derived from it. When the APA-Application closes an SA on an A- PA- or KD-Port with this reason it is to notify the client that the SA and all SAs derived from it are revoked.

recovery: When an A-, PA- or KD-Client closes an SA with this reason it is because a (recovery) operation on the client has invalidated all the existing SAs of this client. When the APA-Application closes an SA on an A- PA- or KD-Port with this reason it is because all SAs on a port and those derived from them need to be invalidated for recovery reasons.

4.5.2 Results

The CloseSA abstract operation returns an empty result as indication of success.

4.5.3 Errors

The errors that may disrupt the closeSA operation are listed below, and for each abstract error the clause in which it is defined is listed. Note that the success of this operation also depends on the success of the DeclareOperationContext operation with which it is combined. In particular it must identify a valid SA.

Abstract error	Clause
OperationNotSupported	4.7.15
Unspecified	4.7.21

4.6 RevokeCertificate

This abstract operation is used by an A-Server or PA-Server to inform the APA-Client that certain certificates have been revoked. The revoked certificates are identified by a set of criteria and it is the responsibility of the responder to act according to the revocation. The operation is permitted only within an existing Security Association.

```
RevokeCertificate ::= ABSTRACT-OPERATION
    ARGUMENT   RevokeCertificateArgument
    RESULT     RevokeCertificateResult
    ERRORS       {  CertificateTypeNotSupported,
                    InvalidCondition,
                    OperationNotSupported,
                    Unspecified }

RevokeCertificateArgument ::= SEQUENCE {
    certificateType         [0]    CertType,
    conjuncts               [1]    SEQUENCE OF Conjunct}

CertType ::= CHOICE {
    predefinedType          [0]    ENUMERATED {     auc     (1),
                                                    pac     (2),
                                                    guc     (3)},
    otherType               [1]    OBJECT IDENTIFIER}

Conjunct ::= SEQUENCE OF RevocationCondition

RevocationCondition ::= CHOICE {
    certIdentity            [0]    CertIdentityCondition,
    creationTime            [1]    TimeCondition,
    attribute               [2]    AttributeCondition}

CertIdentityCondition ::= SEQUENCE {
    issuerDomain            [0]    Identifier              OPTIONAL,
    issuerName              [1]    Identifier              OPTIONAL,
    serialNumber            [2]    INTEGER                 OPTIONAL }

TimeCondition ::= SEQUENCE{
    after                   [0]    UTCTime,
    before                  [1]    UTCTime                 OPTIONAL} -- a later time than "after"

AttributeCondition ::= SecurityAttribute

RevokeCertificateResult ::= ENUMERATED{
                                      noCertificates              (1),
                                      noDistribution              (2),
                                      potentialDistribution       (3),
                                      distribution                (4) }
```

4.6.1 Arguments

certificateType
Type of the certificate that is to be revoked.

conjuncts
Specifies the criteria for revocation. These criteria are constructed as a sequence of conjunct that all have to be satisfied for a particular certificate if the revocation is for that certificate. This allows for example the specification of revocation of all certificates issued today, **and** to a given set of principals. A conjunct is itself a

sequence of revocation conditions. Each conjunct is considered satisfied if any one of the conditions inside the conjunct is satisfied.

A revocation condition specifies a particular condition that can lead to the revocation of a certificate. There are three types of conditions. Identity conditions refer to the issuer of certificates with different granularity. Time conditions refer to the time certificates have been created, and attribute conditions allow certificates containing specified attributes to be revoked.

certIdentity
This condition can contain one, two or three of the elements (the same set as appears in the Common Contents part of the Generalised Certificate defined in part 2). If only an issuerDomain is given; it means that all certificates issued by Authorities in the specified domain are affected by this revocation condition. If an issuerName is given; it means that the certificates issued by this Authority are affected. Omission of issuerDomain is permitted when there is a well known default which can be assumed, though if needed, an issuerDomain can also be used in conjunction with an issuerName. If a serialNumber is given the issuerName must also be given to avoid ambiguities. This means that only one individual certificate is specified.

creationTime
This condition means that certificates issued between the specified times ("before" must be later than "after"), or simply after the "after" time, if "before" is not present, are affected by the revocation.

attribute
This condition means that certificates that contain an attribute that is of the same type and has the same value as that specified are affected by the revocation.

4.6.2 Results

noCertificates
No certificates known to this responder, or distributed by this responder satisfy the conditions specified.

noDistribution
All the certificates satisfying the revocation conditions have now been revoked; none of them have been distributed to other principals.

potentialDistribution
All the certificates satisfying the revocation conditions have now been revoked locally by the responder, but there may exist copies of one or more of the revoked certificates due to distribution to other principals.

distribution
All the certificates satisfying the revocation conditions have now been revoked locally by the responder, but one or more of the revoked certificates are known to have been distributed to other principals.

4.6.3 Errors

The errors that may disrupt the revoke certificate abstract operation are listed below, and for each abstract error the clause in which it is defined is listed.

Abstract error	Clause
CertificateTypeNotSupported	4.7.4
OperationNotSupported	4.7.15
InvalidCondition	4.7.10
Unspecified	4.7.21

4.7 Abstract errors arising from common operations

This clause defines the abstract errors that may disrupt atomic abstract operations.

4.7.1 AssocAlreadyOpen

This abstract error reports that the Security Association identified in the operation is already open. The AssocAlreadyOpen error has no parameters.

```
AssocAlreadyOpen ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.2 AUCSpecificError

The AUCSpecificError reports that the responder is unable to make use of the AUC specified in the argument of the abstract operation due to one or more invalid fields in the AUC specific contents. The AUCSpecificError has a parameter that indicates the reason for the error. The error also has as optional parameters the attributes that caused the error and the serial number of the invalid certificate.

```
AUCSpecificError ::= ABSTRACT-ERROR
    PARAMETER  SEQUENCE {
      aucProblem     [0]   AUCProblem,
      serialNumber   [1]   INTEGER                           OPTIONAL,
      attributes     [2]   SEQUENCE OF SecurityAttribute     OPTIONAL }

AUCProblem ::= ENUMERATED {
                          incompatibleSyntaxVersion          (1),
                          invalidProtection                      (2),
                          wrongType                              (3),
                          authenticationLevel                    (4),
                          invalidAttribute                       (5),
                          invalidTime                            (6) }
```

aucProblem

incompatibleSyntaxVersion
This problem indicates that the responder is unable to handle the specified syntax version for the AUC specific contents of the certificate provided in the abstract operation argument.

invalidProtection
This reason indicates that the AUC was invalid for this target because none of the protection method groups passed validation.

wrongType
This problem indicates that the responder can not accept the type of AUC (Primary or Secondary Principal) specified in the argument of the abstract operation.

authenticationLevel
This problem indicates that the responder will not accept the AUC specified in the argument of the abstract operation due to an insufficient authentication level.

invalidAttribute
This problem indicates that the responder will not accept the AUC specified in the argument of the abstract operation due to one or more Attributes being unknown or unacceptable to the responder.

invalidTime
This problem indicates that the responder could not accept the AUC specified in the argument of the abstract operation because the current time is not within the time periods specified in the specific contents part of the AUC.

serialNumber
The serial number of the certificate in question. Present only if more than one certificate could be the source of the error, and the aucProblem was not incompatibleSyntaxVersion.

attributes
This identifies the rejected attribute(s). It is present only if the aucProblem was invalidAttributes.

4.7.3 CertificateError

The CertificateError reports that the certificate specified in the argument of the abstract operation is invalid due to one or more of the fields in the common contents structure or the check value. The CertificateError has a parameter for the cause of the error and optionally the serial number of the invalid certificate.

```
CertificateError ::= ABSTRACT-ERROR
    PARAMETER  SEQUENCE {
    certificateProblem            [0]    CertificateProblem ,
    certificateSerialNumber       [1]    INTEGER OPTIONAL }

CertificateProblem ::=  ENUMERATED {
            incompatibleSyntaxVersion       (1),
            issueProblem                    (2),
            timeProblem                     (3),
            invalidCheckValueType           (4),
            invalidCheckValueTarget         (5),
            invalidSeal                     (6),
            invalidSignature                (7),
            invalidSymmetricAlgId           (8),
            invalidAsymmerticAlgId          (9),
            invalidHashAlgId                (10)
            certificateRevoked              (11)
            certificateExpired              (12)}
```

The serial number of the specified certificate is optionally provided by the responder to identify which certificate caused the error when there is more than one certificate in the argument.

The **CertificateProblem** parameter given in the abstract error has the following meanings:

incompatibleSyntaxVersion
This problem indicates that the responder is unable to handle the specified syntax version of a certificate given in the abstract operation argument.

issueProblem
This problem indicates that the certification authority that issued a certificate specified in the argument of the abstract operation is unknown or not trusted by the responder, or that the certificate has been revoked.

timeProblem
This problem indicates that the time at which the certificate is being validated is earlier than the validity period specified in the certificate's common contents.

invalidCheckValueType
This type of check value is not supported

invalidCheckValueTarget
The check value is a seal but the responder is not the target nominated in the seal.

invalidSeal
This problem indicates that verification of the seal on the certificate specified in the argument of the abstract operation failed.

invalidSignature
This problem indicates that verification of the signature on the certificate specified in the argument of the abstract operation failed.

invalidSymmetricAlgId
This problem indicates that the identifier for a symmetric key algorithm specified in the argument of the abstract operation is unacceptable or unknown to the responder.

invalidAsymmetricAlgId
This problem indicates that the identifier for an asymmetric key algorithm specified in the argument of the abstract operation is unacceptable or unknown to the responder.

invalidHashAlgId
This problem indicates that the identifier for a hash function algorithm specified in the argument of the abstract operation is unacceptable or unknown to the responder.

certificateRevoked
The certificate has been revoked.

certificateExpired
This problem indicates that the time at which the certificate is being validated is later than the validity period specified in the certificate's common contents.

4.7.4 CertificateTypeNotSupported

This abstract error reports that the type of certificate nominated in the operation is not supported.

```
CertificateTypeNotSupported ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.5 ConstructTypesNotSupported

This abstract error reports that none of the Key Construction Data types supported by the initiator are supported by the KD-Server. This error has no parameters.

```
ConstructTypesNotSupported ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.6 InitiatorKDSUnknown

This abstract error reports that initiator KD-Server specified in the argument of the abstract operation is not known to the receiving KD-Server.

```
InitiatorKDSUnknown ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.7 InitiatorUnknown

This abstract error reports that initiator specified in the argument of the abstract operation is not known to the receiving KD-Server.

```
InitiatorUnknown ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.8 InsufficientAuthorisation

The InsufficientAuthorisation error reports that although the authorisation was usable at the server in question, the requested operation has been failed on an access control check.

```
InsufficientAuthorisation ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.9 InvalidAttributeRequirements

Not now used.

4.7.10 InvalidCondition

The InvalidCondition error reports that the APA-Client is unable to revoke any certificate according to the conditions specified in the argument of the abstract operation because the condition itself rules out any valid certificates at all (e.g. the certificate with serial number X **and** serial number Y). The invalid-condition error has no parameters.

```
InvalidCondition ::= ABSTRACT-ERROR
    PARAMETER      serialNumber    INTEGER
```

4.7.11 InvalidDialogueKeyBlock

The InvalidDialogueKeyBlock error reports that the Dialogue Key Block received is unacceptable. There is one parameter indicating the reason.

```
InvalidDialogueKeyBlock ::= ABSTRACT-ERROR
    PARAMETER  ENUMERATED{
                        useAlgorithmNotSupported          (1),
                        useHashAlgNotSupported            (2),
                        derivationAlgorithmNotSupported   (3),
                        noBasicKey                        (4)}
```

useAlgorithmNotSupported
Indicates that a reversible algorithm with which a dialogue key is intended to be used is not supported by the responder.

useHashAlgNotSupported
Indicates that a hash algorithm with which a dialogue key is intended to be used is not supported by the responder.

derivationAlgorithmNotSupported
Indicates that a one way function using which a dialogue key is to be derived is not supported by the responder.

noBasicKey
It is not possible to accept a dialogue key block because there is no basic key from which to derive it.

4.7.12 InvalidKeyBlock

The InvalidKeyBlock error reports that the key block is unacceptable. The InvalidKeyBlock error has one parameter that indicates the reason for the error.

```
InvalidKeyBlock ::= ABSTRACT-ERROR
    PARAMETER  ENUMERATED{
                        kdSchemeNotSupported     (1),
                        invalidTargetPart        (2),
                        userCertificateError     (3) }
```

The parameter value given in this abstract error has the following meaning:

kdSchemeNotSupported
This reason indicates that the target does not support the Key Distribution scheme specified in the target key block.

invalidTargetPart
This reason indicates that the target can not handle the target part of the key block for reasons that are specific to the key distribution scheme.

userCertificateError

The key block was in the form of a user certificate which was not valid.

4.7.13 KdSchemeNotSupported

This abstract error reports that the Key Distribution Scheme for which Keying Information is required is not supported by the KD-Server.

```
KdSchemeNotSupported ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.14 KiTypeNotSupported

This abstract error reports that the type of Keying Information specified as being required in the argument is not supported by the KD-Server.

```
KiTypeNotSupported ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.15 OperationNotSupported

This abstract error reports that the operation is not supported in this implementation of the Standard.

```
OperationNotSupported ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.16 PACSpecificError

The PAC-specific-error reports that the responder is unable to make use the PAC specified in the argument of the abstract operation due to one or more invalid fields in the PAC specific content. The PAC-specific-error has a parameter that indicates the reason for the error. The error also has as an optional parameter the security attribute(s) that caused the error, and the serial number of the invalid certificate.

```
PACSpecificError ::= ABSTRACT-ERROR
    PARAMETER  SEQUENCE {
        pacProblem          [0]    PACProblem,
        serialNumber        [1]    INTEGER                              OPTIONAL,
        securityAttributes  [2]    SEQUENCE OF SecurityAttribute        OPTIONAL}

PACProblem ::= ENUMERATED {
    historyNotAcceptable            (1),
    incompatibleSyntaxVersion       (2),
    invalidMiscAttributes           (3),
    invalidPrivilegeAttributes      (4),
    invalidProtection               (5),
    invalidTime                     (6),
    invalidTraceLink                (7),
    restrictionTypeNotSupported     (8),
    wrongType                       (9)}
```

pacProblem

historyNotAcceptable
This problem indicates that the responder is unable to accept one or more of the Certificate Identities in the pacHistory field of the PAC specified in the argument of the abstract operation.

incompatibleSyntaxVersion
This problem indicates that the responder is unable to handle the specified syntax version for the PAC specific content of the certificate provided in the abstract operation argument.

invalidProtection
This reason indicates that one or more of the protection methods specified in the PAC failed.

invalidMiscAttributes
This problem indicates that the responder will not accept the PAC specified in the argument of the abstract operation because one or more of the miscellaneous attributes in the PAC are unknown or unacceptable to the responder.

invalidPrivilegeAttributes
This problem indicates that the responder will not accept the PAC specified in the argument of the abstract operation because one or more Privilege Attributes in the PAC are unknown or unacceptable to the responder.

invalidTime
This problem indicates that the responder could not accept the PAC specified in the argument of the abstract operation because the current time is not within the time periods specified in the specific contents part of the PAC.

invalidTraceLink
This problem indicates that the responder will not accept the PAC specified in the argument of the abstract operation because traced delegation is in operation (as described in part 2), and the traceLink in the PAC does not point at the next PAC in the chain.

restrictionTypeNotSupported
The restriction identified in the PAC is a mandatory one, but it is not understood or it is not accepted at this target.

wrongType
This problem indicates that the responder can not accept the type of PAC (Primary, tempered Secondary or untempered Secondary Principal) specified in the argument of the abstract operation.

serialNumber
The serial number of the certificate in question. Present only if more than one certificate could be the source of the error, and the pacProblem was not incompatibleSyntaxVersion.

securityAttributes
This identifies the rejected attribute(s). It is present only if the pacProblem was invalidPrivilegeAttributes or invalidMiscAttributes.

4.7.17 SAfailure

This abstract error reports that an incompatible or unknown SA identifier has been specified.

```
SAFailure ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.18 TargetUnknown

This abstract error reports that the target for the Keying Information specified in the argument of the abstract operation is not known.

```
TargetUnknown ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.19 TimingFailure

This abstract error reports that the operation has failed due to an invalid timestamp or conversation sequence number in the replay protection fields of the argument.

```
TimingFailure ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.20 TrustInformationFailure

This abstract error reports that the APA-Application was unable to provide any trust information.

```
TrustInformationFailure ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

4.7.21 Unspecified

This abstract error reports that the responder is unwilling or unable to give the reason for the failure of the abstract operation.

```
Unspecified ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

5 Authentication Port atomic operations

This clause defines operations supported by the A-Port of the APA Application.

As well as the specific operations described in this clause, the A-Port supports all of the common operations described in clause 4. These are:

• OpenSA	-	see 4.1
• DeclareOperationContext	-	see 4.2
• GetKI	-	see 4.3
• ProcessKI	-	see 4.4
• CloseSA	-	see 4.5
• RevokeCertificate	-	see 4.6

5.1 Authenticate

This abstract operation supports the authentication of Primary and Secondary Principals. Successful authentication will result in the return of an AUC. If the PA-Server for the authenticating principal is collocated with the A-Server, the AUC does not need to be protected against the client since it is only for use by the client. Successful authentication can optionally result in the formation of an SA under whose protection subsequent operations can be conducted. It is the server which first nominates an SA identifier, though because multiple exchanges may be necessary (using Continue Authentication operations) before such an SA can be established, an authentication identifier nominated by the principal can be used if required to link the authentication operations together. SAs are established only in response to successful completion of the authentication exchanges, and their identification may therefore appear instead in the response to a ContinueAuthentication operation. The response in which the SA identifier appears is the first message protected under the SA key that has been established for that SA. A timestamp in the response can be used to prevent its replay. No message sequence number is required since this is always message number one.

Keying Information can also be returned for use in establishing an SA under which subsequent requests for PACs can be authorised to a collocated or separate PA-Port.

A default PAC can be returned if the PA-Server for the authenticating principal is collocated with the A-Server.

```
Authenticate ::= ABSTRACT-OPERATION
    ARGUMENT    AuthenticateArgument
    RESULT          AuthenticateResult
    ERRORS          {   AUCSpecificError,
                        CertificateError,
                        InvalidAuthenticationMethod,
                        InvalidServiceRequested,
                        OperationNotSupported,
                        PACSpecificError,
                        UnacceptableLogonName,
                        Unspecified }

AuthenticateArgument ::= SEQUENCE{
    authenticationParams    [0]    SEQUENCE OF AuthenticationInfo,
    logonName               [1]    PrintableString                  OPTIONAL,
    serviceRequested        [2]    ServiceRequested                 OPTIONAL,
    SAIdentifier            [3]    OCTET STRING                     OPTIONAL,
    ppInfo                  [4]    PrincipalInfo                    OPTIONAL}
                                                     -- for syntax see 4.1
```

```
AuthenticationInfo ::= SEQUENCE {
        authenticationMethod     [0]     AuthenticationMethod,
        exchangeAI               [1]     AuthMparm}

AuthenticationMethod ::= OBJECT IDENTIFIER  -- See Annex B.

AuthMparm ::= CHOICE{
        printableValue           [0]     PrintableString,
        integerValue             [1]     INTEGER,
        octetValue               [2]     OCTET STRING,
        bitStringValue           [3]     BIT STRING,
        otherValue               [4]     ANY}      -- defined by authenticationMethod

ServiceRequested ::= SEQUENCE {
        referralSupported        [0]     BOOLEAN                         OPTIONAL,
        establishSA              [1]     BOOLEAN                         OPTIONAL}

AuthenticateResult ::= SEQUENCE{
        progress                 [0]     ENUMERATED {
                                                 complete  (1),
                                                 continue  (2),
                                                 failed    (3) },
        resultMessage            [1]     CHOICE {
                             completeAuthResult   [0] CompleteAuthResult,
                             continueAuthResult   [1] ContinueAuthResult}    OPTIONAL}

CompleteAuthResult ::= SEQUENCE{
        aServerInformation       [0]     SEQUENCE{
                          aServerKI      [0]     CHOICE {
                                                 keySet          [0]     KeySet
                                                 initKeyBlock    [1]     InitiatorKeyBlock}
                                                                         OPTIONAL,
                          aServerDKB     [1]     DialogueKeyBlock                OPTIONAL,
                                                                 -- defined in part 2
                             auc         [2]     GeneralisedCertificate },
        pServerInformation          [1]     SEQUENCE{
                             pServerKI      [0]     KeySet                    OPTIONAL,
                             pServerAd      [1]     Identifier                OPTIONAL,
                             defaultPAC     [2]     GeneralisedCertificate    OPTIONAL},
        kdServerKI                  [2]         KeySet                        OPTIONAL,
        SAId                        [3]         OCTET STRING                  OPTIONAL,
        timestamp                   [4]         UDT                           OPTIONAL,
        credentialsExpiryWarning    [5]         Printable String              OPTIONAL,
        clientInformation           [6]         String                        OPTIONAL,
        userInformation             [7]         Printable String              OPTIONAL}
```

```
ContinueAuthResult ::= SEQUENCE{
      serverAuthExchangeId       [0]    INTEGER,
      authenticationParams       [1]    SEQUENCE OF AuthenticationInfo}
```

5.1.1 Arguments

authenticationParams
This Standard defines a framework within which specific authentication information choices can be made. Annex B offers some illustrative examples of Authentication Information types. All of these types fit within the general syntactic constructs below:

AuthenticationInfo
This construct defines an element of authentication information. It does so by specifying the method with which the information is associated (authenticationMethod), followed by the actual value associated with it (exchangeAI). The authentication process can involve multiple exchanges of multiple different occurrences of these elements (see Annex B).

authenticationMethod
Identifies the method of authentication being used. The methods described in Annex B are:

for human users:	**for application entities:**
password	
encrypted password	password bits
encrypted replay protected password	symmetric crypto
hashed password	asymmetric crypto
hashed replay protected password	application name only
password used as a key	
passive token	
active token	
voice print	
signature	
fingerprint	
retina pattern	
user certificate	
a zero knowledge method	
location based	

exchangeAI
The value of the authentication element. This is defined in a way which permits efficient implementation of common constructs while at the same time being open ended via the "ANY" construct.

logonName
A name understandable to a human user that the user optionally gives when he wants to authenticate. If the authentication method already includes this identity, the field may be omitted.

serviceRequested
This construct specifies service types that are requested. This include whether the Primary Principal supports referral or not and if the principal requests a stateful or a stateless service (i.e. if an SA has to be established). If this argument is not present, the services offered are determined by the A-Service local policy.

SAIdentifier
Some forms of authentication support the automatic establishment of an SA on successful completion of authentication. The client may use this field to specify an identifier along the same lines as in OpenSA.

ppInfo
This construct qualifies the Primary Principal in the form of a certificate (an AUC or a PAC) with its related protection information. This argument may impact the authentication's acceptance from the authentication server depending on constraints that may be imposed on Secondary Principals when using particular Primary Principals, according to security policy. The construct is further described in 4.1.

5.1.2 Results

progress
Indicates the current status of the authentication process.

– complete :	The authentication is complete and was successful.
– continue :	The authentication is not finished and another exchange is necessary.
– Failed:	The authentication failed.

resultMessage
Body of the authenticate result returned by the A-Server containing arguments for complete authentication and incomplete authentication. This is not present when authentication has failed.

completeAuthResult
Argument for complete authentication :

aServerInformation
Security information in the form of :

- **aServerKI**
optional Keying Information to protect future operations on the A-Server or for establishing an SA with the A-Server. When the A-Server is stateful, only the **initKeyBlock** is returned here, when it is stateless, the full **keySet** is returned,
- **aServerDKB**
optional dialogue key block, to establish a confidentiality dialogue key, and,
- **auc**
an Authentication Certificate. If the Secondary Principal's default PA-Server is collocated with the A-Server, the AUC does not need to be signed or sealed since it only for use by the client.

The general form of Keying Information used for this is described in part 2.

pServerInformation
Security information in the form of Keying Information to protect operations on the default PA-Server or for establishing an SA with the default PA-Server for the principal, the PA-Server's location/address/name (this Standard does not dictate the nature of the address information), and optionally a default PAC. The default PAC is possible only when the principal's A- and PA-Servers are collocated. The form of the Keying Information used for this is described in part 2.

NOTE
The pServerAd part of this argument is intended for use by clients which do not know which PA-Server to use or how to get there.

kdServerKI
This result is returned when a separate KD-Port is used and when the configuration of the APA-Application requires the A-Server to return Keying Information for use in establishing a basic key with the KD-Server.

SAId
The identifier to be used afterwards by the authentication client to refer to this SA. See clause 4.1.1 for the handling of the value of this field. It is a context handle: It refers to a context in the authentication server where information to secure the SA is stored (e.g. the SA key). This argument is absent when the A-Server is stateless.

timestamp
Used to protect the authentication client from a replayed response. Only required when no protection is already provided (either by the network or by a Primary Principal SA when authenticating a Secondary Principal).

credentialsExpiryWarning
Information intended to be displayed as received to warn an authenticating Secondary Principal human user that the credentials being used for authentication are about to expire.

clientInformation
Policy-specific information for use by the client. For policies in which inactivity time-out values for principals are decided by the authentication authority, this field is intended to be used to transmit them to the Subject Sponsor.

userInformation
Information intended to be displayed as received to authenticating Secondary Principals who are human users. It is intended to carry such information as date and time of last logon.

continueAuthResult
Argument for incomplete authentication :

serverAuthExchangeId
This is the A-Server's reference for the authentication exchanges involved in this authentication. This value will be used by the client in subsequent Continue Authentication requests that are part of the same authentication process.

authenticationParams
This authentication information is used by the authentication server to give back to the authentication client authentication arguments according to the authentication method used. (see also 5.2).

5.1.3 Errors

The errors that may disrupt the Authenticate abstract operation are listed below, and for each abstract error the clause in which it is defined is listed. Not that the failure to authenticate is returned as a result, not an error.

Abstract error	Clause
AUCSpecificError	5.9.1
CertificateError	5.9.2
InvalidAuthenticationMethod	5.9.4
InvalidServiceRequested	5.9.9
OperationNotSupported	5.9.10
PACSpecificError	5.9.11
UnacceptableLogonName	5.9.12
Unspecified	5.9.14

5.2 ContinueAuthentication

This abstract operation allows the implementation of authentication schemes of arbitrary complexity. The formal arguments are largely the same as for the Authenticate operation except for the LogonIdentity and the primaryPrincipal attributes which are not part of the request in a ContinueAuthentication operation. The semantics of the information being transferred in a ContinueAuthentication operation or in its results, is dependent on the authentication method used in the preceding Authenticate operation.

```
ContinueAuthentication ::= ABSTRACT-OPERATION
    ARGUMENT    ContinueAuthenticationArgument
    RESULT      AuthenticateResult
    ERRORS         {   InvalidAuthenticationMethod,
                       InvalidExchangeId,
                       OperationNotSupported,
                       Unspecified }

ContinueAuthenticationArgument    ::= SEQUENCE{
    authenticationParams      [0]     SEQUENCE OF AuthenticationInfo,
    serverAuthExchangeId      [1]     INTEGER}
```

5.2.1 Arguments

authenticationParams
See Authenticate above.

serverAuthExchangeId
The value returned from the previous response from the A-Server in this authentication process. Used to link this operation with the previous one.

5.2.2 Results

AuthenticateResult is defined above in 5.1.2.

5.2.3 Errors

The errors that may disrupt the Continue Authentication abstract operation are listed below, and for each abstract error the clause in which it is defined is listed.

Abstract error	Clause
InvalidAuthenticationMethod	5.9.4
InvalidExchangeId	5.9.6
OperationNotSupported	5.9.10
Unspecified	5.9.14

5.3 ChangePassword

This abstract operation allows principals to change their password. There are choices relating to the method of transmitting and choosing the password.

```
ChangePassword ::= ABSTRACT-OPERATION
    ARGUMENT   ChangePasswordArgument
    RESULT     ChangePasswordResult                          OPTIONAL
    ERRORS       {  InsufficientAuthorisation,
                    InvalidAuthenticationMethod,
                    InvalidNewPassword,
                    InvalidOldPassword,
                    OperationNotSupported,
                    UnacceptableLogonName,
                    Unspecified }

ChangePasswordArgument ::= SEQUENCE{
    logonName               [0]   STRING                     OPTIONAL,
    newAuthenticationInfo   [1]   CHOICE{
                                    suppliedPassword   [0]   PasswordInfo,
                                    requestAList       [1]   BOOLEAN},
    oldAuthenticationInfo   [2]   AuthenticationInfo         OPTIONAL}
                                      -- For syntax, see 5.1

ChangePasswordResult   ::= SEQUENCE {
    passwordList          [0] PasswordList   OPTIONAL,
    serverExchangeId      [1] INTEGER        OPTIONAL }
```

```
PasswordInfo    ::= SEQUENCE{
        value                   [0]     BIT STRING,
        algorithmId             [1]     OBJECT IDENTIFIER                   OPTIONAL,
                                                    --NULL for non encrypted password
        encryptionKeyRef        [2]     ENUMERATED {
                                                basicKey        (1),
                                                oldAI           (2)}    }

PasswordList    ::= SEQUENCE{
        values                  [0]     SEQUENCE OF BIT STRING,
        algorithmId             [1]     OBJECT IDENTIFIER                   OPTIONAL,
                                                    -- NULL for non encrypted passwords
        encryptionKeyRef        [2]     ENUMERATED {
                                                basicKey  (1),
                                                oldAI     (2)}
        keyId                   [3]     INTEGER                             OPTIONAL}
```

5.3.1 Arguments

logonName
The change may be specific to a logon name, in which case it must be specified for which logon name the change is to be requested.

newAuthenticationInfo
If the principal is allowed to choose the new AI, new authentication information is specified. If principals can request a list of passwords from which to choose one the request-a-list argument is set to TRUE. Otherwise, the **suppliedPassword** information includes :

- **value**
  The value of the (optionally encrypted) new password,
- **algorithmId**
  The identifier of the algorithm used to encrypt the new password. If NULL is specified here, the new password is not encrypted, if this field is omitted, the default algorithm for Security Information encryption set by the security policy applies here.
- **encryptionKeyRef**
  When the password is encrypted, the reference of the key used for encryption. It can be either :
  – the basic key shared with the A-Server,
  – the old authentication information.

oldAuthenticationInfo
Optionally the old authentication information may also be supplied for additional security. Only a password based authentication method can be specified here.

5.3.2 Results

The result will be returned only if the requestAlist boolean is true.

passwordList
This field of result contains a list of candidate passwords, to be responded to with a ContinueChangePassword operation. The rules of the suppliedPassword field of the argument apply to the list of passwords in the result.

serverExchangeId
Only present when a ContinueChangePassword operation is needed. This is the A-Server's reference for the change password exchanges involved in this change password process. This value will be used by the client in the subsequent ContinueChangePassword request that is part of the same process.

5.3.3 Errors

The errors that may disrupt the Change Password abstract operation are listed below, and for each abstract error the clause in which it is defined is listed.

Abstract error	Clause
InsufficientAuthorisation	5.9.3
InvalidAuthenticationMethod	5.9.4
InvalidNewPassword	5.9.7
InvalidOldPassword	5.9.8
UnacceptableLogonName	5.9.12
Unspecified	5.9.14

5.4 ContinueChangePassword

This abstract operation allows principals to complete a password change when the request-a-list choice has been specified in the previous operation argument.

```
ContinueChangePassword ::= ABSTRACT-OPERATION
    ARGUMENT    ContinueChangePwdArgument
    RESULT
    ERRORS          {   InvalidChoice,
                        InvalidExchangeId,
                        OperationNotSupported,
                        Unspecified }

ContinueChangePwdArgument   ::= SEQUENCE {
    choiceFromList          [0]     INTEGER,
    serverExchangeId        [1]     INTEGER     OPTIONAL }
```

5.4.1 Arguments

choiceFromList
This argument specifies the index number of the chosen password from the sequence of passwords offered in the response to an earlier ChangePW operation with the serverExchangeId specified in the next argument.

serverExchangeId
The value returned in the earlier ChangePW response from the A-Server. Used to link this operation with the previous one in the same change password process.

5.4.2 Results

The Continue Change Password abstract operation returns an empty result as indication of success.

5.4.3 Errors

The errors that may disrupt the Continue Change Password abstract operation are listed below, and for each abstract error the clause in which it is defined is listed.

Abstract error	Clause
InvalidChoice	5.9.5
InvalidExchangeId	5.9.6
OperationNotSupported	5.9.10
Unspecified	5.9.14

5.5 Check Authentication Certificate (CheckAUC)

This abstract operation is used to validate the time validity and revocation status of an AUC, and optionally its check value.

```
CheckAUC ::= ABSTRACT-OPERATION
      ARGUMENT    CheckAUCargument
      RESULT      CheckAUCResult
      ERRORS         {   InsufficientAuthorisation,
                         OperationNotSupported,
                         Unspecified }

CheckAUCArgument ::= CHOICE {  inputAUC        [0]     Generalised Certificate,
                                   inputAUCId   [1]     CertificateId}

CheckAUCResult ::= SEQUENCE{
      answer                   [0]    ENUMERATED{
                                          validAtTimeOfCheck       (1),
                                          syntaxVersionNot Supp    (2),
                                          signingAuthorityNotRec   (3),
                                          aucIdNotKnown            (4),
                                          revoked                  (5),
                                          invalidCheckValue        (6),
                                          invalidValidityPeriod    (7),
                                          invalidAUCTimePeriod     (8),
                                          invalidSyntax            (9) } ,
      timeOfCheckorRevoke      [1]    UTCTime}
```

5.5.1 Arguments

The **CheckAUCArgument** permits either the whole AUC or an identifier for the AUC to be supplied:

inputAUC
inputAUCId

The argument specifies which AUC is to be checked, by presenting either the AUC itself or, if the AUC was created by the A-Server, the AUC's identification. If the whole AUC is presented, the AUC's check value will be checked.

5.5.2 Results

The CheckAUC abstract operation returns the result of the check, and the time when the check took place.

answer
The answer may have one of the following values:

- **validAtTimeofCheck:** The AUC has been successfully checked and found valid,
- **syntaxVersionNotSupp:** The syntax version field in the certificate common contents, or in the AUC specific contents indicates a version not supported by the responder.
- **signingAuthorityNotRec;** The signing authority for the AUC is not recognised by the responder,
- **aucIdNotKnown:** An AUC Identifier has been provided, but it was not one that is known by the responder,
- **revoked:** The AUC has been revoked. The timeOfCheckorRevoke argument indicates the revocation time,
- **invalidCheckValue:** the AUC's check value is invalid. This is may be a sign of a change of content of the AUC, i.e. the integrity check of the AUC shows an integrity violation,
- **invalidValidityPeriod:** the time of the check is outside the times specified in the common contents validity period.
- **invalidAUCTimePeriod:** the time of the check is outside the AUC specific validity time period(s),

- **invalidSyntax:** The AUC could not be parsed. It does not conform to the syntax defined for an AUC.
- **timeOfCheckOrRevoke**

  timeOfCheckOrRevoke is either the time that the AUC validation was made, or if the AUC has been revoked, the revocation time.

5.5.3 Errors

The errors that may disrupt the CheckAUC abstract operation are listed below, and for each abstract error the clause in which it is defined is listed. Note that no AUC-specific errors are returned since this operation has such errors in its normal return value.

Abstract error	Clause
InsufficientAuthorisation	5.9.3
OperationNotSupported	5.9.10
Unspecified	5.9.14

5.6 ConfirmPresence

This abstract operation is used by a stateful A-Server to confirm that a named Secondary Principal is active at a Primary Principal. The operation can be used to control multiple logon sessions by the same Secondary Principal.

```
ConfirmPresence ::= ABSTRACT-OPERATION
    ARGUMENT   ConfirmPresenceArgument
    RESULT     ConfirmPresenceResult
    ERRORS       {  InsufficientAuthorisation,
                    OperationNotSupported,
                    UnknownIdentity,
                    Unspecified }

ConfirmPresenceArgument  ::=   AuthenticatedIdentity

AuthenticatedIdentity    ::=   SecurityAttribute

ConfirmPresenceResult    ::=   ENUMERATED {   present        (1),
                                              inactive       (2),
                                              absent         (3) }
```

5.6.1 Arguments

The AuthenticatedIdentity identifies the principal for which the confirmation of presence is requested. This operation only applies to Secondary Principals and it is the responsibility of the Primary Principal that contains the APA-Client to decide whether or not it considers the Secondary Principal to be present. How this verification takes place is a local matter for the Primary Principal.

5.6.2 Results

present
The principal is present and active.

inactive
The principal is logged on, but is inactive.

absent
The principal is not known

NOTE
If no response is obtained from the APA-Client, for example if the connection is broken, the A-Server should treat this failure as if the principal concerned was not present.

5.6.3 Errors

The errors that may disrupt the ConfirmPresence abstract operation are listed below, and for each abstract error the clause in which it is defined is listed.

Abstract error	Clause
InsufficientAuthorisation	5.9.3
OperationNotSupported	5.9.10.
Unspecified	5.9.14

5.7 Get Authentication Server Name (GetASName)

This abstract operation enables clients to obtain the name of a Secondary Principal's A-Server. It is used when the Primary Principal has been authenticated and when the Secondary Principal's A-Server name is not known to the Primary Principal. The A-Server's name can then be used to obtain Keying Information for communicating with the Secondary Principal's A-Server.

```
GetASName ::= ABSTRACT-OPERATION
    ARGUMENT    GetASNameArgument
    RESULT      GetASNameResult
    ERRORS         {   InsufficientAuthorisation,
                       InvalidService Requested,
                       OperationNotSuported,
                       UnacceptableLogonName,
                       Unspecified }

GetASNameArgument ::= SEQUENCE {
    logonName            [0]     PrintableString,
    referralSupported    [1]     BOOLEAN                     OPTIONAL}

GetASNameResult ::= CHOICE {
    aServerNames         [0]     SEQUENCE OF Identifier,
    referralServerAd     [1]     Identifier }
                                             -- Identifier is defined in part 2
```

5.7.1 Arguments

logonName
Logon name of the principal for whom the A-Server name is requested.

referralSupported
This field indicates to the A-Server whether it is worthwhile returning a referralServerAd.

5.7.2 Results

aServerNames
Names of the Secondary Principal's A-Servers for use as the targetName in a GetKI operation. More than one name may be supplied if the A-Service supports multiple A-Servers for a principal for resilience purposes. This Standard does not lay down any precedence rules for the subsequent use of these names.

referralServerAd
Address of an A-Server that is able to service the GetASName operation. This field is provided only if referral is supported.

5.7.3 Errors

The errors that may disrupt the GetASName abstract operation are listed below, and for each abstract error the clause in which it is defined is listed.

Abstract error	Clause
InsufficientAuthorisation	5.9.3
InvalidServiceRequested	5.9.9
OperationNotSupported	5.9.10
UnacceptableLogonName	5.9.12
Unspecified	5.9.14

5.8 Get Authentication Ticket (GetAT)

This abstract operation enables clients to obtain an AUC or other type of authentication ticket. It can be performed only in the context of a previous successful authentication.

```
GetAT ::= ABSTRACT-OPERATION
    ARGUMENT    GetATArgument
    RESULT      GetATResult
    ERRORS         {  InsufficientAuthorisation,
                      OperationNotSupported,
                      UnacceptableTicketRequirements,
                      Unspecified }

GetATArgument ::= TicketRequirements

GetATResult ::= SEQUENCE {
    aTicket                 [0]    AuthenticationTicket,
    externalControlValues   [1]    ECV                        OPTIONAL}     -- defined in part 2

    AuthenticationTicket ::= CHOICE {
      auc                   [0]    GeneralisedCertificate,                  -- defined in part 2
      other                 [1]    ANY }                 -- defined by TicketRequirements
```

NOTE
The value of authenticationLevel in an AUC will depend on the method(s) of authentication used, according to prevailing security policy.

```
TicketRequirements ::= SEQUENCE{
    typeOfCertificate     [0]    CertificateType,
    protectionType        [1]    ENUMERATED {  seal   (1),
                                               sign   (2)}     OPTIONAL,
    targettingInfo        [2]    Identifier                    OPTIONAL,
                                       -- Identifier is defined in part 2.
    timePeriods           [3]    TimePeriods                   OPTIONAL,
                                       -- TimePeriods is defined in part 2
    certificateControls   [4]    SEQUENCE OF MethodGroup       OPTIONAL}
```

```
CertificateType ::= CHOICE {
    ecmaType           [0]    ENUMERATED {  auc                    (1),
                                               pac                   (2),
                                               proxyOnlyPAC          (3),
                                               initiatorOnlyPAC      (4)},
    otherType          [1]    OBJECT IDENTIFIER}
```

5.8.1 Arguments

This operation has no authorisation argument. Authorisation information is obtained either directly from the OpenSA or Declare Operation Context operation with which it is always combined, or from a previously established Security Association within which the operation is issued. Authorisation failures can still however occur, so an authorisation failure error is appropriate.

The argument takes the form of a **TicketRequirements** construct. This construct is used in a number of operations to specify what kind of return information is required. The term "ticket" is used here to indicate that the extended syntax of CertificateType permits options encompassing symmetrically sealed tickets as well as certificates proper.

typeOfCertificate
For this operation, in the CertificateType field, only the options of ecmaType = auc, or otherType are permitted. The GetACT operation is used for obtaining access control tickets.

protectionType
Indicates how the ticket is to be integrity protected.

targettingInfo
Identifies the target application, if there is a specific one, for which the ticket is required. This argument is used by the A-Server in the provision of a seal for sealed tickets, but may also affect the values carried in the ticket.

timePeriods
Specifies the required time or times during which the ticket is valid. When the ticket is a Generalised Certificate, this argument will cause an appropriate change to the validity field in the Common Contents part of the certificate, and if needed, the Time Periods field in the Specific Contents part.

certificateControls
A set of values to be included in the ticket as protection method groups. The syntax and semantics of **MethodGroup** are defined in part 2. It is used by the client to specify the protection methods required in the ticket being obtained. It is used only for ecmaType ticket requests.

5.8.2 Results

aTicket
This is the proof of authentication returned by the A-service. It is either in the form of an AUC as specified in part 2 of this Standard, or can be of a type not defined in this Standard, but specified in the Ticketrequirements by identifying an "otherType" for the CertificateType argument of the abstract operation.

externalControlValues
This is an optional ECV construct that is associated with the AUC in the result. It is not returned unless the associated ticket type is an AUC. The ECV is defined in part 2 of this Standard.

5.8.3 Errors

The errors that may disrupt the Get Authentication Ticket abstract operation are listed below, and for each abstract error the clause in which it is defined is listed.

Abstract error	Clause
InsufficientAuthorisation	5.9.3
OperationNotSupported	5.9.10
UnacceptableTicketRequirements,	5.9.13
Unspecified	5.9.14

5.9 Abstract errors arising from A-Port operations

This clause defines the errors that may disrupt the authentication-port abstract operations.

5.9.1 AUCSpecificError

This abstract error is the same as that specified in 4.7.

5.9.2 CertificateError

This abstract error is the same as that specified in 4.7.

5.9.3 InsufficientAuthorisation

This abstract error is the same as that specified in 4.7.

5.9.4 InvalidAuthentication method

The invalid-authentication-method error reports that the responder is unable to support the authentication methods specified in the argument of the abstract operation. The invalid-authentication-method abstract error has zero or more parameters that give the invalid authentication method identifier(s).

```
InvalidAuthenticationMethod ::= ABSTRACT-ERROR
    PARAMETER  SEQUENCE OF MethodId OPTIONAL
```

Authentication methods may be invalid either because the A-Server does not support them at all, or policy may specify that a certain method is not available for a certain principal. These reasons are policy dependent and are not reported by this error. If the responder does not want to indicate the particular method that caused the error, the error need have no parameter.

5.9.5 InvalidChoice

The InvalidChoice error reports that the choice specified in the argument of the abstract operation is out of range. The InvalidChoice abstract error has no parameters.

```
InvalidChoice ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

5.9.6 InvalidExchangeId

The InvalidExchangeId error reports that the continued authentication of a principal has failed because serverAuthExchangeId was invalid. The invalidExchangeId abstract error has no parameters.

```
InvalidExchangeId ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

5.9.7 InvalidNewPassword

The new password presented is invalid. There is one parameter indicating the reason.

```
InvalidNewPassword ::= ABSTRACT-ERROR
PARAMETER    ENUMERATED {
    reusedTooSoon                (1),
    tooCloseToExisting           (2),
    tooShort                     (3),
    wrongCharacterMix            (4),
    unspecified                  (5) }
```

reusedTooSoon
The new password has already been used by the principal, and the number of new passwords he has used since then is considered to be too low by the security policy to authorise the reuse of this password for this principal.

tooCloseToExisting
The new password does not have sufficient characters differing from the old one to satisfy the password control policy.

tooShort
The new password is too short to satisfy the password control policy.

wrongCharacterMix
The new password does not have the required mix of alphabetic and other character types to satisfy the password control policy.

unspecified
The password is invalid for reasons unspecified here.

5.9.8 InvalidOldPassword

The password presented in order to authorise the operation is invalid.

```
InvalidOldPassword ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

5.9.9 InvalidServiceRequested

This abstract error reports that the responder is not able to support the service type requested in the argument of the abstract operation. The invalid-service-requested abstract error has no parameters.

```
InvalidServiceRequested ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

5.9.10 OperationNotSupported

This abstract error is the same as that specified in 4.7.

5.9.11 PACSpecificError

This abstract error is the same as that specified in 4.7

5.9.12 UnacceptableLogonName

The unacceptableLogonName error reports that the A-Server is unable to support authentication of the logon name specified in the argument of the abstract operation. The unacceptable-logon-name abstract error has no parameters.

```
UnacceptableLogonName ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

5.9.13 UnacceptableTicketRequirements

The UnacceptableTicketRequirements error reports that the APA-Application is not able to or is unwilling to provide the kind of certificate specified. The UnacceptableTicketRequirements error has one parameter that gives the reason for the error.

```
UnacceptableTicketRequirements ::= ABSTRACT-ERROR
PARAMETER    ENUMERATED {
    unacceptableCertificateType          (1),
    unacceptableProtectionType           (2),
    unacceptableTargetInformation        (3),
    unacceptableCertificateControls      (4) }
```

The parameter value given in the abstract error has the following meaning.

unacceptableCertificateType
The type of certificate (AUC, PAC or other) can not be provided by the APA-Application.

unacceptableProtectionType
The type of integrity protection requested for a certificate (seal or signature) can not be provided by the APA-Application.

unacceptableTargetInformation
The target specified for subsequent use of a certificate is not known to the APA-Application.

unacceptableCertificateControls
One or more of the protection methods specified for a certificate can not be provided or they can not be combined as requested.

5.9.14 Unspecified

This abstract error is the same as that specified in 4.7.

6 Privilege Attribute Port atomic operations

This clause defines operations supported by the PA-Port of the APA Application.

As well as the specific operations described in this clause, the PA-Port supports all of the common operations described in clause 4. These are:

• OpenSA	-	see 4.1
• DeclareOperationContext	-	see 4.2
• GetKI	-	see 4.3
• ProcessKI	-	see 4.4
• CloseSA	-	see 4.5
• RevokeCertificate	-	see 4.6

6.1 Get Access Control Ticket (GetACT)

This abstract operation enables clients to obtain a PAC or other types of access control ticket.

```
GetACT ::= ABSTRACT-OPERATION
    ARGUMENT   GetACTArgument
    RESULT     GetACTResult
    ERRORS        {  InsufficientAuthorisation,
                     OperationNotSupported,
                     UnacceptableAttributeRequirements,
                     UnacceptableRestrictionRequirements,
                     UnacceptableTicketRequirements,
                     Unspecified }

GetACTArgument ::= SEQUENCE{
    ticketReqs              [0]   TicketRequirements            OPTIONAL,
                                          -- for syntax see 5.8
    attributeReqs           [1]   AttributeRequirements         OPTIONAL,
    restrictionReqs         [2]   SEQUENCE OF Restriction       OPTIONAL}
                                          -- for syntax see part 2
    kdServerKiRequired      [3]   BOOLEAN                       OPTIONAL}

AttributeRequirements ::= SEQUENCE{
    attributeSetReferences  [0]   SEQUENCE OF SecurityAttribute OPTIONAL,
    specificAttrRequirements [1]  SEQUENCE OF SecurityAttribute OPTIONAL}
```

```
GetACTResult ::= SEQUENCE {
    accessControlTicket     [0]    CHOICE{
                                   ecmaPAC       [0]    CertandECV, -- defined in part 2
                                   other         [1]    ANY},
                                          -- defined by TicketRequirements.CertificateType.Other
    kdServerKI              [1]    KeySet                              OPTIONAL}
```

6.1.1 Arguments

ticketReqs
Specifies the kind of access control ticket requested and the protection requested for it. Its syntax is given in 5.8. For the typeOfCertificate sub-field, the type: ecmaType with one of the values pac, proxyOnlyPAC and initiatorOnlyPAC, or the type: otherType are the only ones permitted in this operation. Only tickets of type ecmaType are defined in this Standard. "ecmaType.pac" is the default that is assumed if this field is omitted. The targettingInfo sub-field may be set to indicate the target AEF for a sealed PAC.

attributeReqs
Privilege Attributes which the client considers should act as a limitation on those to be provided in the returned ticket. The client provides this information when it does not require its full access capabilities, and wishes to obtain a ticket whose Privilege Attributes are limited in number and/or value. If this argument is omitted, it is assumed that no additional qualification is required. If it is present, only the Privilege Attributes specified will be present. If a specified attribute in the argument is provided with a null value field, the value allocated in the PAC will be the maximum available to the client according to the prevailing security policy, otherwise the value specified will be that returned. If an explicitly nominated value clashes with policy an error is reported and no ticket is returned.

restrictionReqs
Specifies the restrictions requested to be put in the PAC. When the signedExternal field of the required restriction has a NULL value, the PA-Server is requested to generate a public / private key pair as specified in part 2 of this Standard. Otherwise, this field contains a public key. Neither an authorisation check, nor a consistency check is made by the PA-Server on the restriction required.

kdServerKiRequired
When a separate KD-Port is used and when the configuration of the APA-Application requires the PA-Server to be able to return Keying Information, this field specifies whether Keying Information for use in establishing a basic key with the KD-Server is required or not.

6.1.2 Results

This field contains the result when the request has been successfully handled by the PA-Server.

accessControlTicket
The access control ticket is either in the form of a sealed or signed PAC, or a kind of ticket not defined in this Standard.

ecmaPAC
This result is returned when the ticket requested is a PAC as defined in part 2 of this Standard. In this case the result contains the PAC, along with the External Control Values optionally associated with the PAC.

other
Access control ticket, which is not an ECMA PAC, and the type of which is defined by otherType in typeOfCertificate in the ticketRequirement argument.

kdServerKI
This result is returned when the field kdServerKiRequired is present in the argument and set to true. It contains Keying Information for use in establishing a basic key with the KD-Server.

6.1.3 Errors

The errors that may disrupt the GetACT abstract operation are listed below, and for each abstract error the clause in which it is defined is listed.

Abstract error	Clause
InsufficientAuthorisation	6.4.3
UnacceptableAttributeRequirements	6.4.6
UnacceptableRestrictionRequirements	6.4.7
UnacceptableTicketRequirements	6.4.8
Unspecified	6.4.9

6.2 Check Privilege Attribute Certificate (CheckPAC)

This abstract operation is used to validate the time validity and revocation status of a PAC, and optionally its check value.

```
CheckPAC ::= ABSTRACT-OPERATION
    ARGUMENT    CheckPACargument
    RESULT      CheckPACResult
    ERRORS         {  InsufficientAuthorisation,
                      OperationNotSupported,
                      Unspecified }

CheckPACArgument ::= CHOICE {  inputPAC      [0]    Generalised Certificate,
                               inputPACId    [1]    CertificateId}

CheckPACResult ::= SEQUENCE{
    answer                    [0]    ENUMERATED{
                                        validAtTimeOfCheck      (1),
                                        syntaxVersionNot Supp   (2),
                                        signingAuthorityNotRec  (3),
                                        pacIdNotKnown           (4),
                                        revoked                 (5),
                                        invalidCheckValue       (6),
                                        invalidValidityPeriod   (7),
                                        invalidPACTimePeriod    (8),
                                        invalidSyntax           (9) } ,
    timeOfCheckorRevoke       [1]    UTCTime}
```

6.2.1 Arguments

The **CheckPACargument** permits either the whole PAC or an identifier for the PAC to be supplied:

inputPAC
inputPACId

The argument specifies which PAC is to be checked, by presenting either the PAC itself or, if the PAC was created by the PA-Server, the PAC's identification. If the whole PAC is presented, the PAC's check value will be checked.

6.2.2 Results

The CheckPAC abstract operation returns the result of the check, and the time when the check took place.

answer
The answer may have one of the following values:

- **validAtTimeofCheck:** The PAC has been successfully checked and found valid,
- **syntaxVersionNotSupp:** The syntax version field in the certificate common contents, or in the PAC specific contents indicates a version not supported by the responder.

- **signingAuthorityNotRec;** The signing authority for the PAC is not recognised by the responder,
- **pacIdNotKnown:** A PAC Identifier has been provided, but it was not one that is known by the responder,
- **revoked:** The PAC has been revoked. The timeOfCheckorRevoke argument indicates the revocation time,
- **invalidCheckValue:** the PAC's check value is invalid. This is may be a sign of a change of content of the PAC, i.e. the integrity check of the PAC shows an integrity violation,
- **invalidValidityPeriod:** the time of the check is outside the times specified in the common contents validity period.
- **invalidPACTimePeriod:** the time of the check is outside the PAC specific validity time period(s),
- **invalidSyntax:** The PAC could not be parsed. It does not conform to the syntax defined for a PAC.

timeOfCheckOrRevoke
The timeOfCheckOrRevoke is either the time that the PAC validation was made, or if the PAC has been revoked, the revocation time.

6.2.3 Errors

The errors that may disrupt the CheckPAC abstract operation are listed below, and for each abstract error the clause in which it is defined is listed. Note that no PAC-specific errors are returned since this operation has such errors in its normal return value.

Abstract error	Clause
InsufficientAuthorisation	6.4.3
OperationNotSupported	6.4.4
Unspecified	6.4.9

6.3 Refine Privilege Attribute Certificate (RefinePAC)

This abstract operation is used to derive a PAC from an already possessed PAC. The new PAC will have less scope or strength than the original.

```
RefinePAC ::= ABSTRACT-OPERATION
    ARGUMENT    RefinePACArgument
    RESULT      RefinePACResult
    ERRORS          {   CertificateError,
                        InsufficientAuthorisation,
                        OperationNotSupported,
                        PACSpecificError,
                        UnacceptableAttributeRequirements,
                        UnacceptableTicketRequirements,
                        Unspecified }

RefinePACArgument ::= SEQUENCE {
    pac                     [0]     GeneralisedCertificate,
    pacControls             [1]     ECV,
    pacRefinementInfo       [2]     PACRefinementInfo}

PACRefinementInfo    ::=    SEQUENCE{
    ticketReqs              [0]     TicketRequirements                  OPTIONAL,
    attributeReqs           [1]     AttributeRequirements               OPTIONAL}
```

```
RefinePACResult        ::=     SEQUENCE{
     refinedPAC                [0]     Generalised Certificate,
     newPACControls            [1]     ECV                                    OPTIONAL}
```

6.3.1 Arguments

pac
The PAC to be refined.

pacControls
Information proving the caller is a valid user of the PAC. The ECV construct is defined in part 2.

pacRefinementInfo
This is used to request particular values for the contents of the refined PAC. TicketRequirements is defined in 5.8. AttributeRequirements is defined in 6.1.

6.3.2 Results

The return of a result indicates the successful refinement of the PAC specified in the argument of the operation. The result carries two arguments: the requested PAC and optionally **newPACControls** an External Control Value needed to use the PAC. When newPACControls is present, the control values it contains are freshly generated to avoid two different PACs from the same PA-Server being protected by the same ECV. The replacements correspond only to the ECVs presented in the ECV argument. Method groups corresponding to ECVs not presented in the ECV argument are omitted.

6.3.3 Errors

The errors that may disrupt the refine PAC abstract operation are listed below, and for each abstract error the clause in which it is defined is listed.

Abstract error	Clause
CertificateError	6.4.2
InsufficientAuthorisation	6.4.3
OperationNotSupported	6.4.4
PACSpecificError	6.4.5
UnacceptableAttributeRequirements	6.4.6
UnacceptableTicketRequirements	6.4.8
Unspecified	6.4.9

6.4 Abstract errors arising from PA-Port operations

This clause defines the errors that may disrupt the privilege attribute port abstract operations.

6.4.1 AUCSpecificError

This abstract error is the same as that specified in 4.7.

6.4.2 CertificateError

This abstract error is the same as that specified in 4.7.

6.4.3 InsufficientAuthorisation

This abstract error is the same as that specified in 4.7.

6.4.4 OperationNotSupported

This abstract error is the same as that specified in 4.7.

6.4.5 PACSpecificError

This abstract error is the same as that specified in 4.7.

6.4.6 UnacceptableAttributeRequirements

The UnacceptableAttributeRequirements error reports that the APA-Application is unable to provide the Privilege Attributes specified in the argument of the abstract operation because of a policy violation or inconsistency with Primary Principal information.

```
UnacceptableAttributeRequirements ::= ABSTRACT-ERROR
    PARAMETER    ENUMERATED {
    unacceptableRefinement          (1),
    unacceptableAttributeSet        (2),
    unacceptableAttributeValues     (3) }
```

unacceptableRefinement
This problem indicates that the APA-Application will not provide a PAC containing the privilege attributes specified in the argument of the abstract operation. No information about the reason for the error is given.

unacceptableAttributeSet
This problem indicates that the APA-Application will not provide a PAC containing the privilege attributes specified in the argument of the abstract operation. The reason is that the set of privilege attributes is inconsistent with policy.

unacceptableAttributeValues
This problem indicates that the APA-Application will not provide a PAC containing the privilege attributes specified in the argument of the abstract operation. The reason is that the actual privilege attribute values requested are inconsistent with policy.

6.4.7 UnacceptableRestrictionRequirements

The Unacceptable-Restriction-Requirements error reports that the APA-Application is unable to generate a public / private key pair for the algorithm identifier specified in the argument of the abstract operation by the algId field in restrictionReqs. The error has one parameter that carries the algorithm identifier that caused the error.

```
UnacceptableRestrictionRequirements ::= ABSTRACT-ERROR
    PARAMETER    algId    OBJECT IDENTIFIER
```

6.4.8 UnacceptableTicketRequirements

This abstract error is the same as that specified in 5.9.

6.4.9 Unspecified

This abstract error is the same as that specified in 4.7.

7 Key Distribution Port atomic operations

This clause defines operations supported by the KD-Port of the APA Application.

All of the Key Distribution Port operations are available also through the A-Port and PA-Port, and are therefore described as common operations in clause 4. The common operations supported via the KD-Port are:

• OpenSA	-	see 4.1
• DeclareOperationContext	-	see 4.2
• GetKI	-	see 4.3
• ProcessKI	-	see 4.4
• CloseSA	-	see 4.5

8 Combined operations

The previous clauses defined simple atomic operations from which combined operations may be constructed. The complete list of atomic operations is given below. An abbreviation for each, used as described below is given.

Combination of operations is often mandatory since the Security Context within which an operation is to be issued has to be established in order to provide authorisation and/or protection of the operation. The security context is either dynamically established or referred to by the OpenSA or DeclareOperationContext atomic operations. Bind and Unbind are unprotected, do not require authorisation and cannot be combined with any other operations.

Common Operations

OpenSA	(Open)	
DeclareOperationContext	(Declare)	-- not permitted alone
GetKI	(GetKI)	-- not permitted alone
ProcessKI	(ProcessKI)	-- not permitted alone
CloseSA	(Close)	-- not permitted alone
Revoke Certificate	(Revoke)	-- not permitted alone

Authentication port operations

Authenticate	(Auth)	
ContinueAuthentication	(ContAuth)	
ChangePassword	(ChangePW)	-- not permitted alone
ContinueChangePassword	(ContChangePW)	-- not permitted alone
CheckAUC	(CheckAUC)	
ConfirmPresence	(Confirm)	-- not permitted alone
GetASName	(GetASName)	-- not permitted alone
GetAT	(GetAT)	-- not permitted alone

Privilege Attribute port operations

GetACT	(GetACT)	-- not permitted alone
CheckPAC	(CheckPAC)	
RefinePAC	(RefinePAC)	

In this clause, the ways in which the above atomic operations are permitted to be combined are specified. The combinations permitted differ from port to port.

A combined operation is constructed by concatenating firstly the arguments and secondly the returns of its atomic components, each in the form of a SEQUENCE. There is one ERROR argument for the whole combined operation. So the combined operation XandYandZ has a construction:

```
XandYandZ ::= ABSTRACT-OPERATION
    ARGUMENT        XandYandZArgument
    RESULT          XandYandZResult
    ERRORS      {    -- those applicable from the constituent atomic operations
                    }

XandYandZArgument ::=   SEQUENCE{       Xargument,
                                        Yargument,
                                        Zargument}

XandYandZResult ::=     SEQUENCE{       Xresult,
                                        Yresult,
                                        Zresult}
```

Combined operations are named in the manner shown above, but with abbreviated names for the atomic constituents. The abbreviations are shown in parentheses in the list above.

8.1 Authentication port combined operations

The following combinations are allowed on an Authentication port. They are consumer initiated unless otherwise stated:

- DeclareandAuth
- DeclareandContAuth
- DeclareandChangePW
- DeclareandContChangePW
- DeclareandCheckAUC

- DeclareandConfirm	-- from supplier
- DeclareandGetASName	
- DeclareandRevoke	-- from supplier
- DeclareandGetAT	
- DeclareandGetATandGetKI	
- DeclareandGetKI	
- DeclareandProcessKI	
- DeclareandOpen	
- DeclareandClose	-- from either consumer or supplier
- OpenandAuth	
- OpenandChangePW	
- OpenandCheckAUC	
- OpenandGetASName	
- OpenandGetAT	
- OpenandGetATandGetKI	
- OpenandGetKI	
- OpenandProcessKI	

8.2 Privilege Attribute port combined operations

- DeclareandGetACT	
- DeclareandGetACTandGetKI	
- DeclareandGetKI	
- DeclareandCheckPAC	
- DeclareandProcessKI	
- DeclareandRefinePAC	
- DeclareandOpen	
- DeclareandRevoke	-- from supplier
- DeclareandClose	-- from either consumer or supplier
- OpenandGetACT	
- OpenandGetACTandGetKI	
- OpenandGetKI	
- OpenandCheckPAC	
- OpenandRefinePAC	

8.3 Key Distribution port combined operations

- DeclareandGetKI	
- DeclareandProcessKI	
- OpenandGetKI	
- OpenandProcessKI	
- DeclareandOpen	
- DeclareandClose	-- from either consumer or supplier

Annex A

(Informative)

Information Model

A.1 Information model

The APA-Application makes use of three information stores: one for information related to each of: Authentication services, Privilege Attribute Services and Key Distribution Services. These information stores are part of the SMIB as described in [ISO 7498-2]. They are described in clauses A.2, A.3 and A.4

This Standard identifies the following types of information held in the APA-Application:

- identification information: for any claimant (either a Primary or a Secondary Principal) that requires access to a protected resource,
- verification authentication information (verification AI): information specific to a claimant to allow verification of its claimed identity,
- Authentication Certificate (AUC): supplied on request, by an A-Server as result of a successful authentication,
- privilege attributes of principals,
- Privilege Attribute Certificate (PAC): supplied on request, by a PA-Server,
- Keying Information supplied on request by a KD-Server, either directly or through an A- or PA-Server,
- recovery information; used to initiate recovery operations on the APA-Application,
- management information; used for management and administration of the APA-Application (e.g. specifying events that should be audited),
- auditing information: generated to allow auditing of the actions and operations of the APA-Application.

A.2 Authentication information base

The authentication information base contains information connected with the use, management and recovery of the authentication related operations on the APA-Application. This information forms the basis for the authentication process. This information is defined in terms of managed objects, attributes and attribute packages. The following only gives a summary of these managed objects.

A.2.1 Authentication Profile object class

The Authentication Profile object contains all information related to a particular principal. This information is defined as a set of attributes described below.

A.2.1.1 Identities

A principal has a number of identities associated with it, as follows:

- Logon Name
- Authenticated Identity
- Audit Identity

These are defined in part 1 of this Standard. A principal may have more than one Logon Name.

A.2.1.2 Authentication methods

The data under this general heading defines the different methods available to a Principal, possibly using different associated Logon Names.

A.2.1.3 Authentication state

For stateful A-Servers, the purpose of this attribute type is to record the current condition of the authentication of the principal. There may be more than one instance of it, depending on whether the principal is permitted to be authenticated more than once at the same time. These attributes change during the course of an authentication process. Example values might be:

- logon in progress,
- authenticated or not, and method(s) used,
- times logged-on and logged-off,
- re-authenticating,
- record of certificates issued,
- Keying Information which protects the secure association,
- Security Association identifier.

A.2.1.4 Authentication control

The purpose of this attribute is to provide for the enabling and disabling of authentication for a given Principal. Example values might be:

- disabled and left the system: this Principal is not permitted to authenticate. It is not intended that the Principal will again be enabled.
- disabled temporarily: this Principal is suspended, i.e. may not at present authenticate, but may later be enabled.
- enabled: this Principal is able to authenticate.

A.2.2 Global A-Service data object class

This object contains the global - service level - information held by the Authentication Information Base. It contains the following attributes, which are described without syntax or reference to how they are used.

A.2.2.1 Secret keys of this Authority

This variable contains the authority Secret Key(s) paired with Privilege Attribute Service names, that the Authentication Authority shares with these PA-Servers in order to seal Authentication Certificates.

A.2.2.2 Private keys of this Authority

This variable contains the Private Key(s), used by the Authority to sign Authentication Certificates.

A.2.2.3 User certificates of this Authority

This variable contains one or more user certificates which may be passed to clients (and which may be passed on to other entities) which need to validate Authentication Certificates signed by this Authority.

A.2.2.4 Recognised Authorities

The identities and Public Keys of Authentication and Certification Authorities recognised by this APA-Application.

A.2.2.5 Other APA-Servers

The Identities of and the public keys and secret keys to be used with other APA-Servers in this distributed application. This includes secret keys shared with other A- or PA-Servers for key distribution purposes.

A.3 Privilege Attribute information base

The Privilege Attribute information base contains information connected with the use, management and recovery of Privilege Attribute operations on the APA-Application.

Privilege Attributes are a set of attributes stored in the Privilege Attribute Information base and associated with a given principal. In addition, usage controls such as validity constraints, target qualifier attributes, etc. are provided. How the

Privilege Attribute values and usage controls are established and managed is subject to security policy and outside scope of this ECMA Standard.

The information forms the basis for the process of granting Privilege Attributes. For each principal the Privilege Attribute Information base contains a Principal Access Profile which contains information as described below. In addition, it contains global service level information.

A.3.1 Principal Access Profile class

This object has an instance for each registered principal. It contains attributes associated with principals. The Privilege Attribute services provided by the APA-Application make use of and result in transformation of this profile object. The attributes in the profile are given below.

A.3.1.1 Identities

A principal has a number of identities associated with it, as follows:

- Authenticated Identity
- Audit Identity
- Non-repudiation Identity
- Charging Identifier

These are defined in part 1 of this Standard. They may appear in PACs issued to the principal concerned, for control purposes. They are not used as access privilege attributes (for which see below).

A.3.1.2 Principal Privileges

Privilege Attributes, which are used to determine the access rights of their possessor. One type of Privilege Attribute of special interest is access identity. The purpose of access identity is to provide identity information on which an access control decision can be based. It need not bear any relation to the Principal's Authenticated Identity. Other types of Privilege Attribute are groups, roles, capabilities and clearances.

The issue of privileges may be subject to the following criteria: the Authority providing the AUC, the authentication level indicated in the AUC, and for a Secondary Principal: the Privilege Attributes supplied in the PAC of the Primary Principal. For each criterion, or for some combination of them the following may apply: only some Privilege Attribute types may be issued, only some Privilege Attribute types and values may be issued, only some Attribute Set References (ASRs) may be requested.

A.3.1.3 Principal Access State

For Stateful PA-Servers, this variable records the state of a given principal: active or inactive, the Keying Information which protects the Security Association(s) currently in use for this principal, and the Security Association identifier(s). Other state-related information includes:

- The Authentication Certificate (AUC) used to authorise requests made by this principal.
- A record of privileges currently issued to this principal. It consists of copies of the PAC's issued along with their date and time of issue

A.3.2 Global service data class

This object contains the global - service level - information held by the Privilege Attribute information base.

It contains the following attributes, which are described without syntax or reference to the packages in which they are used.

A.3.2.1 Secret keys of this Authority

This variable contains the authority secret keys paired with target AEF names, that the PA-Server shares with these target AEFs in order to seal the PAC.

A.3.2.2 Private keys of this Privilege Attribute Application

This variable contains the Private Key(s), used by the Authority to sign Privilege Attribute Certificates.

A.3.2.3 User certificates of this Authority

This variable contains one or more certificates which may be passed to Clients (and which may be passed on to other entities) which need to validate certificates signed by this Authority.

A.3.2.4 Recognised Authorities

The identities and Public Keys of Authentication, Privilege Attribute and Certification Authorities recognised by this APA-Application.

A.3.2.5 Other APA-Servers

The Identities of and the public keys and secret keys to be used with other APA-Servers in this distributed application. This includes secret keys shared with other A- or PA-Servers for key distribution purposes.

A.4 Key Distribution information base

The Key Distribution information base contains information connected with the use, management and recovery of the Key Distribution operations on the APA-Application.

The information forms the basis for the process of granting Keying Information. For each target AEF the Key Distribution Information base contains a target AEF profile which contains the information described below. In addition, it contains global service level information.

A.4.1 Target AEF Profile class

This object has an instance for each registered target AEF. It contains Keys and application components associated with target AEF. The Key Distribution services provided by the APA-Application make use of and result in transformation of this profile object. The attributes in the profile are given below.

A.4.1.1 Target AEF identity

Each target AEF has an identity which appears in the Keying Information it issues

A.4.1.2 Target AEF key distribution profile

Key distribution parameters, which are used to determine how to build the key set. This includes :

- Associated application components
- Key distribution scheme supported
- "OtherKeyBlock" type
- Long term secret key shared with the target AEF, if a symmetric key distribution scheme is in use,
- Optionally a Dynamically Established Shared Key with that target AEF.

A.4.2 Global service data class

This object contains the global - service level - information held by the Key Distribution information base.

It contains the following attributes, which are described without syntax or reference to the packages in which they are used.

A.4.2.1 Private keys of this Privilege Attribute Application

This variable contains the Private Key(s), used by the Authority to optionally sign a key set, if the key distribution scheme requires it.

A.4.2.2 User certificates of this Authority

This variable contains one or more certificates which may be passed to clients (and which may be passed on to other entities) which need to validate key set signed by this Authority.

A.4.2.3 Recognised Authorities

The identities and Public Keys of Key Distribution and Certification Authorities recognised by this APA-Application.

A.4.2.4 Other APA-Servers

The Identities of and the public keys and secret keys to be used with other APA-Servers in this distributed application, in the context of some key distribution schemes.

Annex B

(Informative)

Authentication Information Types

This Annex gives examples of specific Authentication Information Types. It categorises them in terms of the principles below, first for the authentication of human users and second for the authentication of application entities. Finally there is a brief discussion of combinations of methods and protocol issues. The principles are:

Human Users
. Principle A - something known
. Principle B - something possessed
. Principle C - immutable characteristic
. Principle D - trusted third party
. Principle E - context

Application Entities
. Principle A - something known
. Principle D - trusted third party
. Principle E - context

B.1 Exchange AI for human users

B.1.1 Principle A - something known

Seven different methods are given for authenticating human users with a password. A password can take three forms, or the password can be used to form a cryptographic key under which exchange-AI is encrypted. Thus different identifiers are required for the methods. A transferred password can either be in the form of a character string or a sequence of reversibly or irreversibly encrypted bytes. If encrypted it can be either protected from replay attack or not. The different methods are identified by the following object identifiers:

```
password            ::=    { id-am 0 }
encryptedPassword   ::=    { id-am 1 }
encryptedRepPwd     ::=    { id-am 2 }
hashedPassword      ::=    { id-am 3 }
hashedRepPwd        ::=    { id-am 4 }
passwordAsKey       ::=    { id-am 5 }
```

If the methodID is "password", the password is exchanged in clear text. The type of the exchange AI is:

```
PasswordAI ::=  PrintableString
```

If the methodID is "encryptedPassword", the password is encrypted before it is exchanged. The type of the exchange AI is:

```
EncPasswordAI ::= SEQUENCE {
       encPassword         [0]    BIT STRING,
       encAlgId            [1]    AlgorithmIdentifier     OPTIONAL,
       keyIdentifier       [2]    INTEGER                 OPTIONAL}
```

If the methodID is "encryptedRepPwd", the password is encrypted before it is exchanged, and the exchange AI is replay protected with a unique number. The type of the exchange AI is:

```
EncRepPwdAI ::= SEQUENCE {
        encPassword         [0]     BIT STRING,
        encAlgId            [1]     AlgorithmIdentifier     OPTIONAL,
        keyIdentifier       [2]     INTEGER                 OPTIONAL,
        uniqueNumber        [3]     UniqueNumber}-- defined in 4.1
```

If the methodID is "hashedPassword", the password is hashed with an optional hash parameter (e.g. the user logon name) before it is exchanged. The type of the exchange AI is:

```
HashedPasswordAI ::= SEQUENCE {
        hashedPassword      [0]     BIT STRING,
        hashParameter       [1]     OCTET STRING            OPTIONAL,
        hashAlgId           [2]     AlgorithmIdentifier     OPTIONAL}
```

If the methodID is "hashedRepPwd", the password is hashed with an optional hash parameter, and with an unique number used to protect the exchange AI against replay. The type of the exchange AI is:

```
HashedRepPwdAI ::= SEQUENCE {
        hashedPassword      [0]     BIT STRING,
        hashParameter       [1]     OCTET STRING            OPTIONAL,
        uniqueNumber        [2]     UniqueNumber,-- defined in 4.1
        hashAlgId           [3]     AlgorithmIdentifier     OPTIONAL}
```

If the methodID is "passwordAsKey", the password is being used as the source of a cryptographic key used to encrypt other data. There are many possibilities for the syntax of exchange AI.

B.1.2 Principle B - something possessed

There are several authentication methods available for this principle. The detailed semantics of the information passed as exchange AI is out of scope of this Annex. The methods are identified by the object identifiers below:

```
passiveToken    ::=     { id-am 6 }
activeToken     ::=     { id-am 7 }
```

Both the passive and active devices communicate device AI in the form of bit strings.

```
DeviceAI        ::=     BIT STRING
```

B.1.3 Principle C - immutable characteristic

There are numerous authentication methods available for measuring immutable characteristics of humans. The detailed description of this information is outside the scope of this Annex, though it does however distinguish between some methods identified by the following object identifiers:

```
voicePrint          ::=     { id-am 8 }
signature           ::=     { id-am 9 }
fingerprint         ::=     { id-am 10 }
retinaPattern       ::=     { id-am 11 }
toeSmell            ::=     { id-am 99 }
```

A voice print is some representation of the characteristics of a human voice. The exchange AI for the voicePrint method is communicated as a bit string:

A handwritten signature is a representation of the actual signature or some characteristic measured during the process of writing the signature. The exchange AI for the signature method is communicated as a bit string.

A fingerprint is a representation of the skin-pattern on one or more of the fingers of the user. The exchange AI for the fingerprint method is communicated as a bit string.

A retina pattern is a representation of certain characteristics of the retina of a user. It is considered unique to each user and is measured by scanning the retina to get an 'imprint' of it. The exchange AI for the retinaPattern method is communicated as a bit string.

So all of these methods have the same syntax:

```
CharacteristicAI     ::=     BIT STRING
```

The toe smell of any person is quite characteristic, and for all intents and purposes can be considered unique. It often takes the form of an odour which can be represented as:

```
ToeSmellAI                  ::=        PHEROMONE STRING
```

B.1.4 Principle D - trusted third party

If a trusted third party has established authentication then some information authorised by it may be passed as exchange AI. This requires the identity of the authority to be known by the APA-Application. The methods described in this Annex are zero-knowledge, and make use of a user certificate from the Directory Standard. There are several different zero knowledge protocols that may be useful depending on security policy. Only one is given here.

```
userCertificate             ::=     { id-am 12 }
zeroKnowledgeMeth1          ::=     { id-am 13 }
```

the user certificate method AI takes the form of:

```
DirectoryAI  ::=   SEQUENCE {
                          certificate       [0]     Certificate,
                          token             [1]     Token}
```

where Certificate is imported from [ISO/IEC 9594] Part 8 and Token is imported from [ISO/IEC 9594] Part 3.

Zero knowledge protocols are identity based authentication protocols which will establish the identity of a user by proving knowledge of a secret but without revealing that secret. The information exchanged during the protocol is specific to each different zero-knowledge protocol. The method identified here is defined as a challenge-response, and is provided to help illustrate the way in which in different exchanges different exchangeAI values may be presented. The protocol exchanges involved are described in B.3.

```
ZeroKnowledgeAI ::=    CHOICE {
     challenge       [0]     BIT STRING,
     response        [1]     BIT STRING}
```

B.1.5 Principle E - context

The context information applicable for this Standard is the location of the user. Hence the method is denoted 'location' and will typically be some form of address of the Primary Principal. The location method is identified by the object identifier below:

Note that using "location" as a security argument can provide only very weak authentication unless dedicated ports are used on the security server. However if such a port is used, then there is no need to put this information in the protocol. The same type of argument can be applied to other context arguments: it is better that the server establish these values, not the client.

```
location              ::=     { id-am 14 }
```

The location method uses the address of a user as exchange AI, but see caveats above.

```
LocationAI            ::=     BIT STRING -- some form of address
```

B.2 Exchange AI for application entities

B.2.1 Principle A - something known

There are three methods described for application entities using this principle. They are identified by the object identifiers below:

```
passwordBits       ::=    { id-am 15 }
symmetricCrypto    ::=    { id-am 16 }
asymmetricCrypto   ::=    { id-am 17 }
```

The symmetric crypto methods that can be used are described in [ISO/IEC 9798-2].

The asymmetric crypto methods that can be used are described in [ISO/IEC 9798-3].

The syntax of the exchange AI is specific to each method.

B.2.2 Principle D - trusted third party

For this principle only the user certificate method applies during authentication of applications. See B.1.4.

B.2.3 Principle E - context

For applications this principle is similar to the location method but is very weak. The method for this is:

```
applicationName    ::=    { id-am 18 }
```

The syntax of the corresponding AI is:

```
ApplicationNameAI ::=    PrintableString
```

B.3 Combinations and Protocol

In principle any of the methods above can be combined, though in practice not all combinations are sensible. The Authenticate operation permits the client either to present multiple AuthenticationInfo elements in one call, or to present further authentication information in a ContinueAuthentication operation following the return of "continue" in the progress field of AuthenticateResult returned from the Authenticate operation. Further exchanges can be requested by the server by ContinueAuthentication also returning a "continue" value.

Challenge-response forms of authentication as defined in B.1.4 are also handled via the ContinueAuthentication operation. For example, in the Authenticate operation the "challenge" BIT STRING can optionally be a challenge to the server, or it could be omitted if no challenge is being offered. In the Authenticate operation the "response" BIT STRING would be omitted. The server returns "continue" along with a challenge in the "authenticationParams" field in Authenticate result, and if challenged itself, a response.

Annex C

(Normative)

ASN.1 - Object Identifier Usage

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

DEFINITIONS ::=
BEGIN

EXPORTS
     module, id-ot, id-pt ;

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

-- Categories of information objects

module       OBJECT IDENTIFIER ::= { apa 1 }
object       OBJECT IDENTIFIER ::= { apa 2 }
port         OBJECT IDENTIFIER ::= { apa 3 }
methods      OBJECT IDENTIFIER ::= { apa 4 }

-- Modules

usefulDefinitions                 OBJECT IDENTIFIER ::= { module 1 }
apaObjectIdentifiers              OBJECT IDENTIFIER ::= { module 2 }
securityInformationObjects        OBJECT IDENTIFIER ::= { module 3 }
apaAbstractService                OBJECT IDENTIFIER ::= { module 4 }
selectedAuthenticationMethods     OBJECT IDENTIFIER ::= { module 5 }

--Synonyms

id-ot    OBJECT IDENTIFIER ::= object
id-pt    OBJECT IDENTIFIER ::= port
id-am    OBJECT IDENTIFIER ::= methods

END -- of useful definitions
```

Annex D

(Normative)

ASN.1 - Object Identifiers

```
ApaObjectIdentifiers {  iso(1) identified-organisation(3) icd-ecma(0012) standard(0)
                        apa(219) modules(1) apaObjectIdentifiers(2) }

DEFINITIONS ::=
BEGIN

EXPORTS
     id-ot-apa-client, id-ot-apa-application,
     id-pt-a, id-pt-pa, id-pt-kd;

IMPORTS
     id-ot, id-pt,
        FROM UsefulDefinitions {
                        iso(1) identified-organisation(3) icd-ecma(0012) standard(0) apa(219)
                        modules(1) usefulDefinitions(1) } ;

--Objects

id-ot-apa-client        OBJECT IDENTIFIER ::= { id-ot 1 }
id-ot-apa-application   OBJECT IDENTIFIER ::= { id-ot 2 }

-- Port types

id-pt-a                 OBJECT IDENTIFIER ::= { id-pt 1 }
id-pt-pa                OBJECT IDENTIFIER ::= { id-pt 2 }
id-pt-kd                OBJECT IDENTIFIER ::= { id-pt 3 }

END -- of ApaObjectIdentifiers
```

Annex E

(Normative)

ASN.1 - Abstract Service

```
ApaAbstractService {   iso(1) identified-organisation(3) icd-ecma(0012) standard(0)
                       apa(219) modules(1) apaAbstractService(4) }
DEFINITIONS ::=

BEGIN
-- exports everything

IMPORTS

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

    OBJECT, PORT, ABSTRACT-BIND, ABSTRACT-UNBIND,
    ABSTRACT-OPERATION, ABSTRACT-ERROR
      FROM AbstractServiceNotation {
                       joint-iso-ccitt mhs-motis(6) asdc(2) modules(0) notation(1) }

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

    DirectoryBindArgument
      -- Note that this is being imported solely for the purpose of importing
      -- its constituent Token construct, which is not separately exported.
      FROM DirectoryAbstractService {
                       joint-iso-ccitt ds(5) modules(1) directoryAbstractService(2) }

    id-ot-apa-client, id-ot-apa-application,
    id-pt-a, id-pt-pa, id-pt-kd
      FROM ApaObjectIdentifiers apaObjectIdentifiers

    OtherKeyBlock, DialogueKeyBlock, CertandECV, ECV, KeySet, Identifier,
    GeneralisedCertificate, TimePeriods, Restrictions, SecurityAttribute
      FROM SecurityInformationObjects securityInformationObjects ;
-- End of imports
-- Objects and ports

APA-Application OBJECT
      PORTS {    authentication-port            [S],
                     privilege-attribute-port    [S],
                     key-distribution-port       [S]}
      ::= id-ot-apa-application

APA-Client OBJECT
      PORTS {    authentication-port            [C],
                     privilege-attribute-port    [C],
                     key-distribution-port       [C]}
      ::= id-ot-apa-client

authentication-port PORT

CONSUMER INVOKES {
      OpenSA,
      Authenticate,
      ContinueAuthentication,
      CheckAUC,
      DeclareandAuth,
      DeclareandContAuth,
      DeclareandChangePW,
      DeclareandContChangePW,
      DeclareandCheckAUC,
      DeclareandGetASName,
      DeclareandGetAT,
      DeclareandGetATandGetKI,
      DeclareandGetKI,
      DeclareandOpen,
      DeclareandClose,
      DeclareandProcessKI,
      OpenandAuth,
      OpenandChangePW,
      OpenandCheckAUC,
      OpenandGetASName,
      OpenandGetAT,
      OpenandGetATandGetKI,
      OpenandGetKI,
      OpenandProcessKI}
```

```
SUPPLIER INVOKES {
        DeclareandConfirm,
        DeclareandRevoke,
        DeclareandClose}
        ::= id-pt-a

privilege-attribute-port PORT

CONSUMER INVOKES {
        OpenSA,
        CheckPAC,
        RefinePAC,
        DeclareandGetACT,
        DeclareandGetACTandGetKI,
        DeclareandGetKI,
        DeclareandCheckPAC,
        DeclareandProcessKI,
        DeclareandRefinePAC,
        DeclareandOpen,
        DeclareandClose,
        OpenandGetACT,
        OpenandGetACTandGetKI,
        OpenandGetKI,
        OpenandCheckPAC,
        OpenandProcessKI,
        OpenandRefinePAC}

SUPPLIER INVOKES {
        DeclareandClose}
        ::= id-pt-pa
```

```
key-distribution-port PORT

CONSUMER INVOKES {
    OpenSA,
    DeclareandGetKI,
    DeclareandProcessKI,
    OpenandGetKI,
    OpenandProcessKI,
    DeclareandOpen,
    DeclareandClose}

SUPPLIER INVOKES {
    DeclareandClose}
    ::= id-pt-kd

-- Bind and Unbind

A-bind ::= ABSTRACT-BIND
TO { authentication-port }
    BIND
    ARGUMENT        AbindArgument
    RESULT          AbindResult
    BIND-ERROR      AbindError

AbindArgument ::= SEQUENCE {
    protocolVersionID    ProtocolVersionID,
    serviceType          ServiceType                        OPTIONAL}

ProtocolVersionID   ::= INTEGER

ServiceType         ::= SEQUENCE {
    referralSupported    [0]    BOOLEAN,
    stateSupported       [1]    ENUMERATED {       stateless    (1),
                                                   stateful     (2),
                                                   both         (3)}
    profileSupported     [2]    OBJECT IDENTIFIER   OPTIONAL}

AbindResult ::= SEQUENCE {
    clientInformation   [0]    STRING                          OPTIONAL,
    serviceType         [1]    ServiceType}

AbindError ::= ENUMERATED {
    busy                            (1),
    serviceTypeUnsupported          (2),
    unspecified                     (3)}
```

```
A-unbind ::= ABSTRACT-UNBIND
FROM { authentication-port }

PA-bind ::= ABSTRACT-BIND
TO { privilege-attribute-port }
      BIND
      ARGUMENT            PAbindArgument
      RESULT              PAbindResult
      BIND-ERROR  PA      bindError

PAbindArgument ::= AbindArgument

PAbindResult ::= AbindResult

PAbindError ::= AbindError

PA-unbind ::= ABSTRACT-UNBIND
FROM { privilege-attribute-port }

KD-bind ::= ABSTRACT-BIND
TO { key-distribution-port }
      BIND
      ARGUMENT        KDbindArgument
      RESULT          KDbindResult
      BIND-ERROR      KDbindError

KDbindArgument ::= SEQUENCE {
      protocolVersionID         ProtocolVersionID,
      kdServiceType             KDserviceType                  OPTIONAL}

KDserviceType       ::= SEQUENCE {
      stateSupported         [1]     ENUMERATED {       stateless      (1),
                                                        stateful       (2),
                                                        both           (3)}
      profilesSupported      [2]     OBJECT IDENTIFIER      OPTIONAL}

KDbindResult ::= ServiceType

KDbindError ::= AbindError

KD-unbind ::= ABSTRACT-UNBIND
FROM { key-distribution-port }
```

```
-- Operations, arguments and results

OpenSA ::= ABSTRACT-OPERATION
    ARGUMENT    OpenSAArgument
    RESULT      OpenSAResult
    ERRORS      {   AssocAlreadyOpen
                    AUCSpecificError,
                    CertificateError,
                    InsufficientAuthorisation,
                    InvalidDialogueKeyBlock,
                    InvalidKeyBlock,
                    OperationNotSupported,
                    PACSpecificError,
                    TimingFailure,
                    Unspecified}

OpenSAArgument ::= SEQUENCE{
    SAIdentifier            [0]     OCTET STRING,
    otherKeyBlock           [1]     OtherKeyBlock,                          -- defined in part 2
    dialogueKeyBlock        [2]     DialogueKeyBlock    OPTIONAL,           -- defined in part 2
    authorisation           [3]     Authorisation,          -- to be associated with the SA
    protection              [4]     Protection          OPTIONAL

Authorisation       ::= SEQUENCE {
    callingPrincipalInfo    [0]     PrincipalInfo       OPTIONAL,
    ppInfo                  [1]     PrincipalInfo       OPTIONAL}

PrincipalInfo       ::= CHOICE {
    aucInfo                 [0]     CertandECV,
    pacInfo                 [1]     CertandECV}
                    -- CertandECV is defined in part 2

Protection          ::= CHOICE {
        timestamp           [0]     UniqueNumber,
        saSeqNumber         [1]     INTEGER VALUE 1}

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

OpenSAResult    ::= SEQUENCE {
        SAIdentifier        [0]     OCTET STRING,
        userInformation     [1]     Printable String    OPTIONAL,
        protection          [2]     Protection          OPTIONAL}
```

```
DeclareOperationContext ::= ABSTRACT-OPERATION
    ARGUMENT        DeclareOperationContextArgument
    RESULT          DeclareOperationContextResult
    ERRORS            {  AUCSpecificError,
                         InvalidKeyBlock,
                         InvalidDialogueKeyBlock,
                         OperationNotSupported,
                         PACSpecificError,
                         SAFailure,
                         TimingFailure,
                         Unspecified }

DeclareOperationContextArgument ::= SEQUENCE{
        operationContext     [0]   OperationContext,
        dialogueKeyBlock     [1]   DialogueKeyBlock                  OPTIONAL,
        authorisation        [2]   Authorisation                     OPTIONAL,
                                       -- Authorisation is defined in 4.1
        protection           [3]   Protection                        OPTIONAL}
                                       -- Protection is defined in 4.1

OperationContext ::= CHOICE {
        saId                 [0]   OCTET STRING,      -- SA ID for stateful operation
        otherKeyBlock        [1]   OtherKeyBlock }  -- keying info for stateless operation

DeclareOperationContextResult ::=  Protection                        OPTIONAL

GetKI ::= ABSTRACT-OPERATION
    ARGUMENT        GetKIArgument
    RESULT          GetKIResult
    ERRORS            {  ConstructTypesNotSupported,
                         InitiatorUnknown,
                         KITypeNotSupported,
                         OperationNotSupported,
                         TargetUnknown,
                         TrustInformationFailure,
                         Unspecified }
```

```
GetKIArgument ::= SEQUENCE {
      kiTypeRequired           [0]     ENUMERATED {   targetKeyBlock        (1),
                                                             ucECMA              (2),
                                                             ucDirectory         (3),
                                                             trustInfo           (4)}
      initiatorName            [1]     Identifier                            OPTIONAL,
      targetName               [2]     Identifier                            OPTIONAL,
      keyConstructSupported    [3]     SEQUENCE OF KeyConstructType          OPTIONAL,
      helpfulInformation       [4]     SEQUENCE OF Certificate               OPTIONAL
                                   -- Certificate is imported from [ISO/IEC 9594-8:1990]}

KeyConstructType::= CHOICE {
      predefinedType           [0]     ENUMERATED {          oneWay          (1),
                                                             encrypted       (2),
                                                             keyIndex        (3) }
      otherType                [1]     OBJECT IDENTIFIER}

GetKIResult::= CHOICE {
      keySet                   [0]     KeySet,               -- defined in part 2
      trustInformation         [1]     TrustedAuthorities}

TrustedAuthorities   ::= SEQUENCE{
      root CA                  [0]     RootCA,
      branchCA                 [1]     CADistinguishedName,
      leafCA                   [2]     CADistinguishedName}

RootCA               ::= SEQUENCE{
      caName                   [0]     CADistinguishedName,
      publicKey Value          [1]     BIT STRING,
      validityperiod           [2]     UTCTime}

ProcessKI ::= ABSTRACT-OPERATION
      ARGUMENT          ProcessKIArgument
      RESULT            ProcessKIResult
      ERRORS               {    InitiatorKDSUnknown,
                                KdSchemeNotSupported,
                                OperationNotSupported,
                                TargetUnknown,
                                Unspecified }

ProcessKIArgument ::= SEQUENCE {
      kdSchemeOID                   [0]     OBJECT IDENTIFIER,
      initiatorKDSname              [1]     Identifier,
      targetKDSpart                 [2]     ANY,      -- Defined by kdSchemeOID. See part 2
      targetAEFName                 [3]     Identifier}
```

```
ProcessKIResult::= SEQUENCE {
      processedKeyBlock             [0]     ANY       -- Defined by kdSchemeOID. See part 2
      initiatorKDSdomain            [1]     Identifier}

CloseSA :: = ABSTRACT-OPERATION
      ARGUMENT    CloseSAArgument
      RESULT
      ERRORS      {    OperationNotSupported,
                       Unspecified}

CloseSAArgument ::=    Reason

Reason      ENUMERATED {        logoff          (1),
                                done            (2),
                                timeout         (3),
                                notPresent      (4),
                                revocation          (5),
                                recovery        (6)}

RevokeCertificate ::= ABSTRACT-OPERATION
      ARGUMENT    RevokeCertificateArgument
      RESULT      RevokeCertificateResult
      ERRORS          {    CertificateTypeNotSupported,
                           InvalidCondition,
                           OperationNotSupported,
                           Unspecified }

RevokeCertificateArgument ::= SEQUENCE {
      certificateType          [0]     CertType,
      conjuncts                [1]     SEQUENCE OF Conjunct}

CertType ::= CHOICE {
      predefinedType           [0]     ENUMERATED {      auc          (1),
                                                         pac          (2),
                                                         guc          (3)},
      otherType                [1]     OBJECT IDENTIFIER}

Conjunct ::= SEQUENCE OF RevocationCondition

RevocationCondition ::= CHOICE {
      certIdentity             [0]     CertIdentityCondition,
      creationTime             [1]     TimeCondition,
      attribute                [2]     AttributeCondition}

CertIdentityCondition ::= SEQUENCE {
      issuerDomain             [0]     Identifier            OPTIONAL,
      issuerName               [1]     Identifier            OPTIONAL,
      serialNumber             [2]     INTEGER               OPTIONAL }
```

```
TimeCondition ::= SEQUENCE{
      after                    [0]     UTCTime,
      before                   [1]     UTCTime              OPTIONAL} -- a later time than "after"

AttributeCondition ::= SecurityAttribute

RevokeCertificateResult ::= ENUMERATED{
                                              noCertificates              (1),
                                              noDistribution              (2),
                                              potentialDistribution       (3),
                                              distribution                (4) }

AssocAlreadyOpen ::= ABSTRACT-ERROR
      PARAMETER  NULL

AUCSpecificError ::= ABSTRACT-ERROR
      PARAMETER  SEQUENCE {
            aucProblem       [0]     AUCProblem,
            serialNumber     [1]     INTEGER                              OPTIONAL,
            attributes       [2]     SEQUENCE OF SecurityAttribute        OPTIONAL }

AUCProblem ::= ENUMERATED {
                                     incompatibleSyntaxVersion        (1),
                                     invalidProtection                (2),
                                     wrongType                        (3),
                                     authenticationLevel              (4),
                                     invalidAttribute                 (5),
                                     invalidTime                      (6) }

CertificateError ::= ABSTRACT-ERROR
      PARAMETER  SEQUENCE {
      certificateProblem              [0]     CertificateProblem ,
      certificateSerialNumber         [1]     INTEGER OPTIONAL }

CertificateProblem ::=  ENUMERATED {
            incompatibleSyntaxVersion    (1),
            issueProblem                 (2),
            timeProblem                  (3),
            invalidCheckValueType        (4),
            invalidCheckValueTarget      (5),
            invalidSeal                  (6),
            invalidSignature             (7),
            invalidSymmetricAlgId        (8),
            invalidAsymmerticAlgId       (9),
            invalidHashAlgId             (10)
            certificateRevoked           (11)
            certificateExpired           (12)}
```

```
CertificateTypeNotSupported ::= ABSTRACT-ERROR
    PARAMETER  NULL

ConstructTypesNotSupported ::= ABSTRACT-ERROR
    PARAMETER  NULL

InitiatorKDSUnknown ::= ABSTRACT-ERROR
    PARAMETER  NULL

InitiatorUnknown ::= ABSTRACT-ERROR
    PARAMETER  NULL

InsufficientAuthorisation ::= ABSTRACT-ERROR
    PARAMETER  NULL

InvalidCondition ::= ABSTRACT-ERROR
    PARAMETER      serialNumber    INTEGER

InvalidDialogueKeyBlock ::= ABSTRACT-ERROR
    PARAMETER  ENUMERATED{
                            useAlgorithmNotSupported            (1),
                            useHashAlgNotSupported              (2),
                            derivationAlgorithmNotSupported     (3),
                            noBasicKey                          (4)}

InvalidKeyBlock ::= ABSTRACT-ERROR
    PARAMETER  ENUMERATED{
                            kdSchemeNotSupported                (1),
                            invalidTargetPart                   (2),
                            userCertificateError                (3) }

KdSchemeNotSupported ::= ABSTRACT-ERROR
    PARAMETER  NULL

KiTypeNotSupported ::= ABSTRACT-ERROR
    PARAMETER  NULL

OperationNotSupported ::= ABSTRACT-ERROR
    PARAMETER  NULL

PACSpecificError ::= ABSTRACT-ERROR
    PARAMETER  SEQUENCE {
        pacProblem          [0]   PACProblem,
        serialNumber        [1]   INTEGER                         OPTIONAL,
        securityAttributes  [2]   SEQUENCE OF SecurityAttribute   OPTIONAL}
```

```
PACProblem ::= ENUMERATED {
    historyNotAcceptable            (1),
    incompatibleSyntaxVersion       (2),
    invalidMiscAttributes           (3),
    invalidPrivilegeAttributes      (4),
    invalidProtection               (5),
    invalidTime                     (6),
    invalidTraceLink                (7),
    restrictionTypeNotSupported     (8),
    wrongType                       (9)}

SAFailure ::= ABSTRACT-ERROR
    PARAMETER  NULL

TargetUnknown ::= ABSTRACT-ERROR
    PARAMETER  NULL

TimingFailure ::= ABSTRACT-ERROR
    PARAMETER  NULL

TrustInformationFailure ::= ABSTRACT-ERROR
    PARAMETER  NULL

Unspecified ::= ABSTRACT-ERROR
    PARAMETER  NULL

Authenticate ::= ABSTRACT-OPERATION
    ARGUMENT   AuthenticateArgument
    RESULT          AuthenticateResult
    ERRORS          {  AUCSpecificError,
                       CertificateError,
                       InvalidAuthenticationMethod,
                       InvalidServiceRequested,
                       OperationNotSupported,
                       PACSpecificError,
                       UnacceptableLogonName,
                       Unspecified }

AuthenticateArgument ::= SEQUENCE{
    authenticationParams      [0]   SEQUENCE OF AuthenticationInfo,
    logonName                 [1]   PrintableString             OPTIONAL,
    serviceRequested          [2]   ServiceRequested            OPTIONAL,
    SAIdentifier              [3]   OCTET STRING                OPTIONAL,
    ppInfo                    [4]   PrincipalInfo               OPTIONAL}
                                                  -- for syntax see 4.1
```

```
AuthenticationInfo ::= SEQUENCE {
    authenticationMethod      [0]   AuthenticationMethod,
    exchangeAI                [1]   AuthMparm}

AuthenticationMethod ::= OBJECT IDENTIFIER  -- See Annex B.

AuthMparm ::= CHOICE{
    printableValue            [0]   PrintableString,
    integerValue              [1]   INTEGER,
    octetValue                [2]   OCTET STRING,
    bitStringValue            [3]   BIT STRING,
    otherValue                [4]   ANY}     -- defined by authenticationMethod

ServiceRequested ::= SEQUENCE {
    referralSupported         [0]   BOOLEAN                         OPTIONAL,
    establishSA               [1]   BOOLEAN                         OPTIONAL}

AuthenticateResult ::= SEQUENCE{
    progress                  [0]   ENUMERATED {
                                          complete     (1),
                                          continue     (2),
                                          failed       (3) },
    resultMessage             [1]   CHOICE {
                        completeAuthResult   [0] CompleteAuthResult,
                        continueAuthResult   [1] ContinueAuthResult}    OPTIONAL}

CompleteAuthResult ::= SEQUENCE{
    aServerInformation        [0]   SEQUENCE{
                        aServerKI      [0]    CHOICE {
                                       keySet          [0]     KeySet
                                       initKeyBlock    [1]     InitiatorKeyBlock}
                                                                OPTIONAL,
                        aServerDKB     [1]    DialogueKeyBlock          OPTIONAL,
                                                             -- defined in part 2
                        auc            [2]    GeneralisedCertificate },
    pServerInformation        [1]   SEQUENCE{
                        pServerKI      [0]    KeySet                    OPTIONAL,
                        pServerAd      [1]    Identifier                OPTIONAL,
                        defaultPAC     [2]    GeneralisedCertificate    OPTIONAL},
    kdServerKI                [2]   KeySet                          OPTIONAL,
    SAId                      [3]   OCTET STRING                    OPTIONAL,
    timestamp                 [4]   UDT                             OPTIONAL,
    credentialsExpiryWarning  [5]   Printable String                OPTIONAL,
    clientInformation         [6]   String                          OPTIONAL,
    userInformation           [7]   Printable String                OPTIONAL}
```

```
ContinueAuthResult ::= SEQUENCE{
    serverAuthExchangeId      [0]    INTEGER,
    authenticationParams      [1]    SEQUENCE OF AuthenticationInfo}

ContinueAuthentication ::= ABSTRACT-OPERATION
    ARGUMENT       ContinueAuthenticationArgument
    RESULT         AuthenticateResult
    ERRORS         {  InvalidAuthenticationMethod,
                      InvalidExchangeId,
                      OperationNotSupported,
                      Unspecified }

ContinueAuthenticationArgument    ::= SEQUENCE{
    authenticationParams   [0]    SEQUENCE OF AuthenticationInfo,
    serverAuthExchangeId   [1]    INTEGER}

ChangePassword ::= ABSTRACT-OPERATION
    ARGUMENT    ChangePasswordArgument
    RESULT      ChangePasswordResult OPTIONAL
    ERRORS         {  InsufficientAuthorisation,
                      InvalidAuthenticationMethod,
                      InvalidNewPassword,
                      InvalidOldPassword,
                      OperationNotSupported,
                      UnacceptableLogonName,
                      Unspecified }

ChangePasswordArgument ::= SEQUENCE{
    logonName                 [0]    STRING                           OPTIONAL,
    newAuthenticationInfo     [1]    CHOICE{
                                     suppliedPassword   [0]     PasswordInfo,
                                     requestAList       [1]     BOOLEAN},
    oldAuthenticationInfo     [2]    AuthenticationInfo               OPTIONAL}
                                          -- For syntax, see 5.1

ChangePasswordResult   ::= SEQUENCE {
    passwordList           [0] PasswordList    OPTIONAL,
    serverExchangeId       [1] INTEGER         OPTIONAL }

PasswordInfo    ::= SEQUENCE{
    value                     [0]    BIT STRING,
    algorithmId               [1]    OBJECT IDENTIFIER                OPTIONAL,
                                                 --NULL for non encrypted password
    encryptionKeyRef          [2]    ENUMERATED {
                                        basicKey        (1),
                                        oldAI           (2)}       }
```

```
PasswordList     ::= SEQUENCE{
      values                  [0]     SEQUENCE OF BIT STRING,
      algorithmId             [1]     OBJECT IDENTIFIER                        OPTIONAL,
                                                    -- NULL for non encrypted passwords
      encryptionKeyRef        [2]     ENUMERATED {
                                                 basicKey (1),
                                                 oldAI      (2)}
      keyId                   [3]     INTEGER                                  OPTIONAL}

ContinueChangePassword ::= ABSTRACT-OPERATION
      ARGUMENT    ContinueChangePwdArgument
      RESULT
      ERRORS          {   InvalidChoice,
                          InvalidExchangeId,
                          OperationNotSupported,
                          Unspecified }

ContinueChangePwdArgument   ::= SEQUENCE {
      choiceFromList        [0]     INTEGER,
      serverExchangeId      [1]     INTEGER     OPTIONAL }

CheckAUC ::= ABSTRACT-OPERATION
      ARGUMENT    CheckAUCargument
      RESULT      CheckAUCResult
      ERRORS          {   InsufficientAuthorisation,
                          OperationNotSupported,
                          Unspecified }

CheckAUCArgument ::= CHOICE {  inputAUC          [0]   Generalised Certificate,
                               inputAUCId        [1]   CertificateId}

CheckAUCResult ::= SEQUENCE{
      answer                  [0]     ENUMERATED{
                                            validAtTimeOfCheck       (1),
                                            syntaxVersionNot Supp    (2),
                                            signingAuthorityNotRec   (3),
                                            aucIdNotKnown            (4),
                                            revoked                  (5),
                                            invalidCheckValue        (6),
                                            invalidValidityPeriod    (7),
                                            invalidAUCTimePeriod     (8),
                                            invalidSyntax            (9) } ,
      timeOfCheckorRevoke     [1]     UTCTime}
```

```
ConfirmPresence ::= ABSTRACT-OPERATION
    ARGUMENT    ConfirmPresenceArgument
    RESULT      ConfirmPresenceResult
    ERRORS          {   InsufficientAuthorisation,
                        OperationNotSupported,
                        UnknownIdentity,
                        Unspecified }

ConfirmPresenceArgument    ::=    AuthenticatedIdentity

AuthenticatedIdentity      ::=    SecurityAttribute

ConfirmPresenceResult      ::=    ENUMERATED {      present (1),
                                                   inactive  (2),
                                                   absent    (3) }

GetASName ::= ABSTRACT-OPERATION
    ARGUMENT    GetASNameArgument
    RESULT      GetASNameResult
    ERRORS      {   InsufficientAuthorisation,
                    InvalidService Requested,
                    OperationNotSuported,
                    UnacceptableLogonName,
                    Unspecified }

GetASNameArgument ::= SEQUENCE {
    logonName           [0]     PrintableString,
    referralSupported   [1]     BOOLEAN                     OPTIONAL}

GetASNameResult ::= CHOICE {
    aServerNames        [0]     SEQUENCE OF Identifier,
    referralServerAd    [1]     Identifier }
                                            -- Identifier is defined in part 2

GetAT ::= ABSTRACT-OPERATION
    ARGUMENT    GetATArgument
    RESULT      GetATResult
    ERRORS      {   InsufficientAuthorisation,
                    OperationNotSupported,
                    UnacceptableTicketRequirements,
                    Unspecified }

GetATArgument ::= TicketRequirements

GetATResult ::= SEQUENCE {
    aTicket                     [0]     AuthenticationTicket,
    externalControlValues       [1]     ECV                 OPTIONAL}     -- defined in part 2
    AuthenticationTicket ::= CHOICE {
      auc                  [0]   GeneralisedCertificate,                -- defined in part 2
      other                [1]   ANY }                  -- defined by TicketRequirements

TicketRequirements ::= SEQUENCE{
    typeOfCertificate      [0]   CertificateType,
    protectionType         [1]   ENUMERATED {   seal     (1),
                                                sign(2)}                OPTIONAL,
    targettingInfo         [2]   Identifier                             OPTIONAL,
                                 -- Identifier is defined in part 2.
    timePeriods            [3]   TimePeriods                            OPTIONAL,
                                 -- TimePeriods is defined in part 2
    certificateControls    [4]   SEQUENCE OF MethodGroup                OPTIONAL}

CertificateType ::= CHOICE {
    ecmaType               [0]   ENUMERATED {  auc                      (1),
                                             pac                        (2),
                                             proxyOnlyPAC               (3),
                                             initiatorOnlyPAC           (4)},
    otherType              [1]   OBJECT IDENTIFIER}

InvalidAuthenticationMethod ::= ABSTRACT-ERROR
    PARAMETER  SEQUENCE OF MethodId OPTIONAL

InvalidChoice ::= ABSTRACT-ERROR
    PARAMETER  NULL

InvalidExchangeId ::= ABSTRACT-ERROR
    PARAMETER  NULL

InvalidNewPassword ::= ABSTRACT-ERROR
PARAMETER    ENUMERATED {
    reusedTooSoon              (1),
    tooCloseToExisting         (2),
    tooShort                   (3),
    wrongCharacterMix          (4),
    unspecified                (5) }

InvalidOldPassword ::= ABSTRACT-ERROR
    PARAMETER  NULL

InvalidServiceRequested ::= ABSTRACT-ERROR
    PARAMETER  NULL

UnacceptableLogonName ::= ABSTRACT-ERROR
    PARAMETER  NULL
```

```
UnacceptableTicketRequirements ::= ABSTRACT-ERROR
PARAMETER    ENUMERATED {
      unacceptableCertificateType           (1),
      unacceptableProtectionType            (2),
      unacceptableTargetInformation         (3),
      unacceptableCertificateControls       (4) }

GetACT ::= ABSTRACT-OPERATION
      ARGUMENT    GetACTArgument
      RESULT      GetACTResult
      ERRORS      {   InsufficientAuthorisation,
                      OperationNotSupported,
                      UnacceptableAttributeRequirements,
                      UnacceptableRestrictionRequirements,
                      UnacceptableTicketRequirements,
                      Unspecified }

GetACTArgument ::= SEQUENCE{
      ticketReqs              [0]    TicketRequirements                OPTIONAL,
                                                 -- for syntax see 5.8
      attributeReqs           [1]    AttributeRequirements             OPTIONAL,
      restrictionReqs         [2]    SEQUENCE OF Restriction           OPTIONAL}
                                                 -- for syntax see part 2
      kdServerKiRequired      [3]    BOOLEAN                           OPTIONAL}

AttributeRequirements ::= SEQUENCE{
      attributeSetReferences     [0]    SEQUENCE OF SecurityAttribute    OPTIONAL,
      specificAttrRequirements   [1]    SEQUENCE OF SecurityAttribute    OPTIONAL}

GetACTResult ::= SEQUENCE {
      accessControlTicket        [0]    CHOICE{
                          ecmaPAC      [0]    CertandECV, -- defined in part 2
                          other        [1]    ANY},
                                 -- defined by TicketRequirements.CertificateType.Other
      kdServerKI                 [1]    KeySet                           OPTIONAL}

CheckPAC ::= ABSTRACT-OPERATION
      ARGUMENT    CheckPACargument
      RESULT      CheckPACResult
      ERRORS   {  InsufficientAuthorisation,
                  OperationNotSupported,
                  Unspecified }

CheckPACArgument ::= CHOICE {  inputPAC     [0]     Generalised Certificate,
                               inputPACId   [1]     CertificateId}
```

```
CheckPACResult ::= SEQUENCE{
    answer                  [0]    ENUMERATED{
                                         validAtTimeOfCheck      (1),
                                         syntaxVersionNot Supp   (2),
                                         signingAuthorityNotRec  (3),
                                         pacIdNotKnown           (4),
                                         revoked                 (5),
                                         invalidCheckValue       (6),
                                         invalidValidityPeriod   (7),
                                         invalidPACTimePeriod    (8),
                                         invalidSyntax           (9) } ,
    timeOfCheckorRevoke     [1]    UTCTime}

RefinePAC ::= ABSTRACT-OPERATION
    ARGUMENT    RefinePACArgument
    RESULT      RefinePACResult
    ERRORS      {  CertificateError,
                   InsufficientAuthorisation,
                   OperationNotSupported,
                   PACSpecificError,
                   UnacceptableAttributeRequirements,
                   UnacceptableTicketRequirements,
                   Unspecified }

RefinePACArgument ::= SEQUENCE {
    pac                     [0]    GeneralisedCertificate,
    pacControls             [1]    ECV,
    pacRefinementInfo       [2]    PACRefinementInfo}

PACRefinementInfo    ::=    SEQUENCE{
    ticketReqs          [0]    TicketRequirements             OPTIONAL,
    attributeReqs       [1]    AttributeRequirements          OPTIONAL}

RefinePACResult      ::=    SEQUENCE{
    refinedPAC          [0]    Generalised Certificate,
    newPACControls      [1]    ECV                            OPTIONAL}

UnacceptableAttributeRequirements ::= ABSTRACT-ERROR
    PARAMETER      ENUMERATED {
    unacceptableRefinement              (1),
    unacceptableAttributeSet            (2),
    unacceptableAttributeValues         (3) }

UnacceptableRestrictionRequirements ::= ABSTRACT-ERROR
    PARAMETER      algId       OBJECT IDENTIFIER
```

```
XandYandZ ::= ABSTRACT-OPERATION
       ARGUMENT    XandYandZArgument
       RESULT         XandYandZResult
       ERRORS              {     -- those applicable from the constituent atomic operations
                                      }

XandYandZArgument ::=  SEQUENCE{         Xargument,
                                                       Yargument,
                                                       Zargument}

XandYandZResult ::=       SEQUENCE{         Xresult,
                                                       Yresult,
                                                       Zresult}

END -- of ApaAbstractService
```

Annex F

(Informative)

ASN.1 - Authentication Methods

```
SelectedAuthenticationMethods {
                    iso(1) identified-organisation(3) icd-ecma(0012) standard(0)
                    apa(219) modules(1) selectedAuthenticationMethods(5) }

DEFINITIONS ::=
BEGIN

-- exports everything

IMPORTS
     id-am
        FROM UsefulDefinitions {
                         iso(1) identified-organisation(3) icd-ecma(0012) standard(0) apa(219)
                         modules(1) usefulDefinitions(1) } ;

-- End of imports

-- Object Identifiers for the authentication methods identified in Annex B of this Standard

password              ::=    { id-am 0 }
encryptedPassword     ::=    { id-am 1 }
encryptedRepPwd       ::=    { id-am 2 }
hashedPassword        ::=    { id-am 3 }
hashedRepPwd          ::=    { id-am 4 }
passwordAsKey         ::=    { id-am 5 }
passiveToken          ::=    { id-am 6 }
activeToken           ::=    { id-am 7 }
voicePrint            ::=    { id-am 8 }
signature             ::=    { id-am 9 }
fingerprint           ::=    { id-am 10 }
retinaPattern         ::=    { id-am 11 }
userCertificate       ::=    { id-am 12 }
zeroKnowledgeMeth1    ::=    { id-am 13 }
location              ::=    { id-am 14 }
passwordBits          ::=    { id-am 15 }
symmetricCrypto       ::=    { id-am 16 }
asymmetricCrypto      ::=    { id-am 17 }
applicationName       ::=    { id-am 18 }

-- Exchange AI Constructs specified in Annex B

PasswordAI ::=   PrintableString
```

```
EncPasswordAI ::= SEQUENCE {
        encPassword         [0]    BIT STRING,
        encAlgId            [1]    AlgorithmIdentifier     OPTIONAL,
        keyIdentifier       [2]    INTEGER                 OPTIONAL}

EncRepPwdAI ::= SEQUENCE {
        encPassword         [0]    BIT STRING,
        encAlgId            [1]    AlgorithmIdentifier     OPTIONAL,
        keyIdentifier       [2]    INTEGER                 OPTIONAL,
        uniqueNumber        [3]    UniqueNumber} -- defined in 4.1

HashedPasswordAI ::= SEQUENCE {
        hashedPassword      [0]    BIT STRING,
        hashParameter       [1]    OCTET STRING            OPTIONAL,
        hashAlgId           [2]    AlgorithmIdentifier     OPTIONAL}

HashedRepPwdAI ::= SEQUENCE {
        hashedPassword      [0]    BIT STRING,
        hashParameter       [1]    OCTET STRING            OPTIONAL,
        uniqueNumber        [2]    UniqueNumber,-- defined in 4.1
        hashAlgId           [3]    AlgorithmIdentifier     OPTIONAL}

DeviceAI            ::=    BIT STRING

CharacteristicAI    ::=    BIT STRING

DirectoryAI  ::=   SEQUENCE {
                           certificate     [0]    Certificate,
                           token           [1]    Token}

ZeroKnowledgeAI ::=   CHOICE {
      challenge       [0]    BIT STRING,
      response        [1]    BIT STRING}

LocationAI          ::=    BIT STRING -- some form of address

ApplicationNameAI      ::= PrintableString


END -- of selectedAuthenticationMethods
```

Annex G

(Informative)

Changes from the first edition (December 1994)

This second edition of Part 3 of Standard ECMA-219 has been changed from the first edition (December 1994) in the following respects:

1. ECMA-138 is now withdrawn as it is largely superseded by this Standard, so the reference to Standard ECMA-138 in Annex A has been removed.
2. An extra argument has been added to the GetKI operation defined in clause 4.3 to permit the caller to transmit to the KD-Server information returned from the target (or obtained from a Directory) to help support the key distribution scheme that is being used. Such information is now available from GSS-API negotiation exchanges that might precede the establishment of the basic key. It is expected that this argument will be especially useful in establishing Security Associations with normal application servers.
3. An extra return has been added to ProcessKI to enable the target AEF to obtain the name of the domain of the initiator's KDS, so that the target AEF can compare the value against the issuer-domain name in the PAC, and ensure that it is equal. This prevents an attack by which a rogue but known external domain can steal PACs from another external domain.
4. The identifier established for Security Associations (in the SAIdentifier argument that appears in OpenSA, DeclareOperationContext and Authenticate operations - see clauses 4.1, 4.2 and 5.1) has been given a new syntax to bring it into line with existing "security context" identifiers in GSS-API implementations. The syntax is now simply OCTET STRING. There is only one of these identifiers per SA, though it may have two components, one chosen by each of the communicating parties.
5. The errors in clause 4.6.3 should have also listed the error: InvalidCondition since it (correctly) appears in the ASN.1. This editing error has now been corrected.
6. Clause 4.7.1 has been clarified to say "Security Association" rather than the less precise "association". The accompanying typo has also been fixed.
7. The error option: timeProblem in CertificateError (see clause 4.7.3) has been split into two parts indicating either that the certificate has expired (i.e. the validity field's notAfter time has been passed), or that the certificate is not yet valid (i.e. the validity field's notBefore time hasn't been reached - see Part 2). This is necessary to give the caller enough information to know how to counter the failure; the implication of expiry is that if another certificate is now obtained, the caller may avoid the failure, since the freshly produced certificate is likely to have a later expiry time, whereas the "too early" error is more likely to require the caller to attempt the action at a later time.
8. An additional error option has also been added to CertificateError to indicate the case of a certificate having been revoked. See clause 4.7.3.
9. The error return: InvalidAttributeRequirements that was defined in clause 4.7.9 has been deleted as it is not used.
10. The otherKeyBlock argument in OpenSA and DeclareOperationContext should have been targetKeyBlock with syntax: TargetKeyBlock. This editing error has now been corrected. The error relating to this argument is now required to cover both of these syntaxes, so has been renamed simply "InvalidKeyBlock" (see clause 4.7.12)
11. The removal of targetKDSpart from the outer syntax of TargetKeyBlock in Part 2 (see Part 2) renders the enumerated option: targetKDSUnknown on clause 4.7.12 unnecessary. It has been removed.