3GPP TSG SA WG3 Security — S3#23 **S3-020421**
9 - 12 July 2002
Helsinki, Finland

---

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020569**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

Title: LS on OSA Security
Response to: n. a.
Release: Rel-5
Work Item: OSA2

Source: CN5
To: SA3
Cc:

Contact Person:
Name: Chelo Abarca (joint API group co-chair), Alcatel
Tel. Number: +33 1 30 77 04 69
E-mail Address: Chelo.Abarca@alcatel.fr

Attachments: N5-020703, N5-020704

---

1. Overall Description:

CN5 would like to thank SA3 for having made a thorough review of the support of security by the OSA API, which has started during the SA3 Bristol meeting, where two CN5 OSA experts were invited to the discussion. As SA3 knows, this kicked-off a detailed discussion on how to solve the OSA security flaws identified by the SA3 experts.

CN5 would like to inform SA3 that this discussion has finally resulted in two CRs to OSA Rel-5, that have been agreed at the current CN5 meeting in Montreal, 8-12 July 2002. These CRs contain changes that are fully in line with the results of the discussion in Bristol. They are attached to this LS for information.

CN5 would like to thank SA3 again for providing the expertise that has allowed improving security support by the OSA API. Security is key for the market acceptance of the OSA API, and we are confident that after this discussion our specification meets our requirements in this area.

2. Actions:

ACTION: No action required.

3. Date of Next CN5 Meetings:

TITLE	TYPE	DATES	LOCATION	CTRY
3GPPCN5#20	WG	23 - 27 Sep 2002	Miami, FLORIDA	US
3GPPCN5#21	WG	28 - 31 Oct 2002	Dublin	IE

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020703**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

CR-Form-v5

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**CRNum**	⌘ **rev**	**-**	⌘	Current version:	**5.0.0**	⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: ⌘	(U)SIM [ ]	ME/UE [ ]	Radio Access Network [ ]	Core Network [X]

Title: ⌘	Add Negotiation of Authentication Mechanism for OSA level Authentication		
Source: ⌘	Chelo Abarca, Alcatel Ultan Mulligan, ETSI		
Work item code: ⌘	OSA2	***Date:*** ⌘	12/07/2002
Category: ⌘	**F**	***Release:*** ⌘	*REL-5*
	Use one of the following categories: ***F*** *(correction)* ***A*** *(corresponds to a correction in an earlier release)* ***B*** *(addition of feature),* ***C*** *(functional modification of feature)* ***D*** *(editorial modification)* Detailed explanations of the above categories can be found in 3GPP TR 21.900.		*Use one of the following releases:* *2 (GSM Phase 2)* *R96 (Release 1996)* *R97 (Release 1997)* *R98 (Release 1998)* *R99 (Release 1999)* *REL-4 (Release 4)* *REL-5 (Release 5)*

Reason for change: ⌘

TS 29.198-3 relies on the use of a challenge-based mechanism (CHAP as per IETF RFC 1994) for authentication of the client application by the framework, and vice-versa. CHAP is chosen as the authentication scheme when the authentication type in the initiateAuthenticate() method is set to P_OSA_AUTHENTICATION.

CHAP requires the support of the MD5 hashing algorithm, as a minimum, with an allowance for support of other algorithms, but no other algorithm is specifically referred to in RFC 1994.

However, since RFC 1994 has been issued, newer, more secure, hashing algorithms have been made available. A mechanism needs to be added to the API to permit negotiation of the hashing algorithm used, in order to take advantage of these newer algorithms.

~~This CR results in from a lengthy communication with SA3 and certain SA3 security experts, who were instrumental in developing the proposals contained in this CR.~~
TS 29.198-3 relies on the use of a challenge-based mechanism (CHAP as per IETF RFC 1994) for authentication of the client application by the framework, and vice-versa. CHAP is chosen as the authentication scheme when the authentication type in the initiateAuthenticate() method is set to P_OSA_AUTHENTICATION.

Because of the lack of detailed reference to RFC 1994 in TS 29.198-3, it is not clear whether CHAP-based OSA authentication must format the challenge and response in packets as described in RFC 1994 or must merely follow the rule given for MD5 processing.

If the Challenge and Response packets as defined in RFC 1994 must be used to format the challenge and the response values, then it is not clear as to what the Name field of the Challenge packet must contain. The Name field must indeed

	be used to identify the sending system. There is no information in the TS as to which value must be put in there. OSA requires that the challenge be encrypted, but this is acknowledged to be of no real value regarding the authentication process. Furthermore, use of encryption introduces complication in that OSA contains no procedures for key exchange, and no instruction as to the use of initialisation vectors and padding algorithms etc. This CR results in from a lengthy communication with SA3 and certain SA3 security experts, who were instrumental in developing the proposals contained in this CR.
Summary of change: ⌘	Add selectAuthenticationMechanism() to IpAPILevelAuthentication interface to permit the client to offer a choice of mechanisms to the Framework. Furthermore deprecate authenticate() and replace it by the a new method challenge(), which does not require encryption of the challenge string. Properly describe the format of the challenge string in the challenge() method. Deprecate selectEncryptionMethod() as encryption of the challenge string will no longer be required. Add extensible types TpAuthMechanism and TpAuthMechanismList to contain the choice of authentication mechanisms. Add an exception in case no acceptable mechanism is available to the Framework.
Consequences if not approved: ⌘	OSA Authentication will be forced to rely on the old MD5 algorithm, which dates from 1992, when newer, more secure alternatives are available. Interoperability problems will result: The format of the challenge and response in the OSA authentication exchange will remain undefined and open to interpretation. Different vendors will implement their own interpretation, forcing application developers to tailor their applications for each vendor's equipement. Since the interoperability problems will be related to the first contact between an application and the Framework, these problems will have a serious impact on the adoption and success of OSA.

Clauses affected: ⌘	6.1.1, 6.3.1.5, 10.3, 11		
Other specs affected: ⌘	☐	Other core specifications ⌘	
	☐	Test specifications	
	☐	O&M Specifications	
Other comments: ⌘			

How to create CRs using this form:

Comprehensive information and tips about how to create CRs can be found at: http://www.3gpp.org/3G_Specs/CRs.htm. Below is a brief summary:

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.

2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.

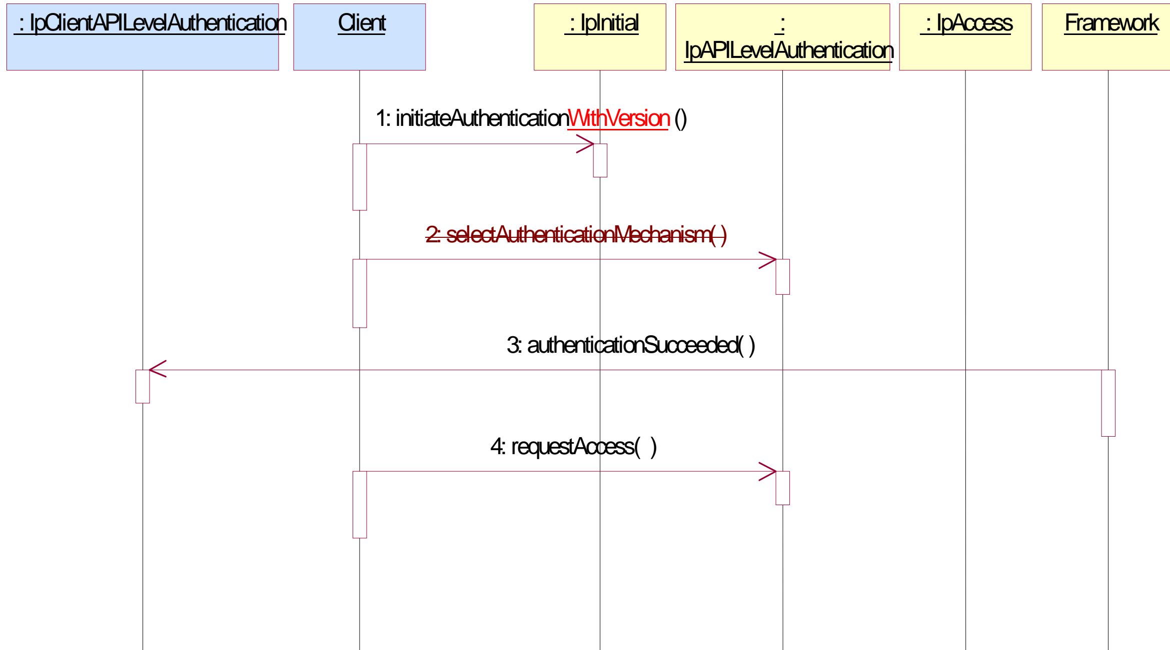
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

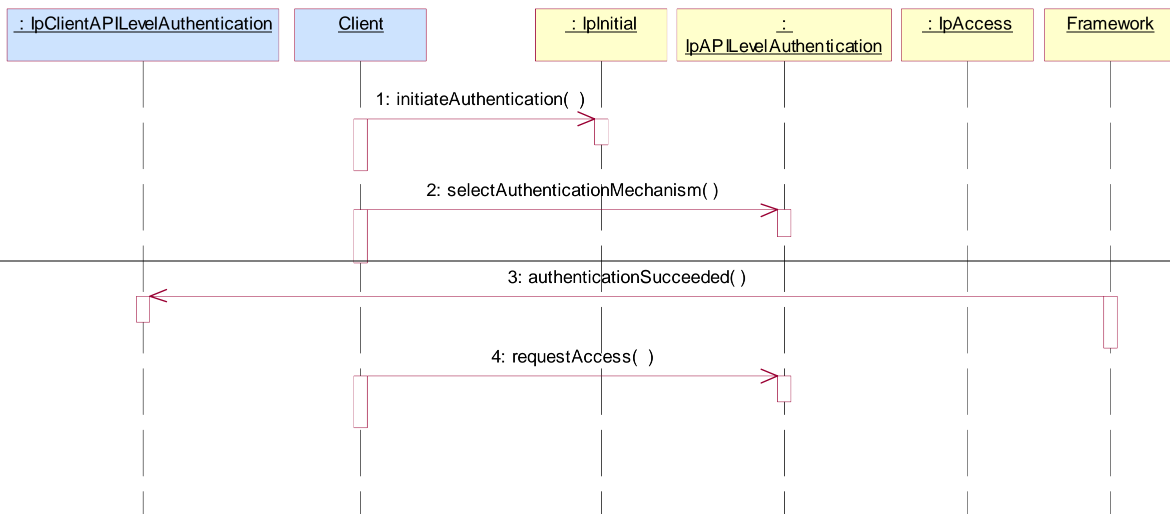
6.1 Sequence Diagrams

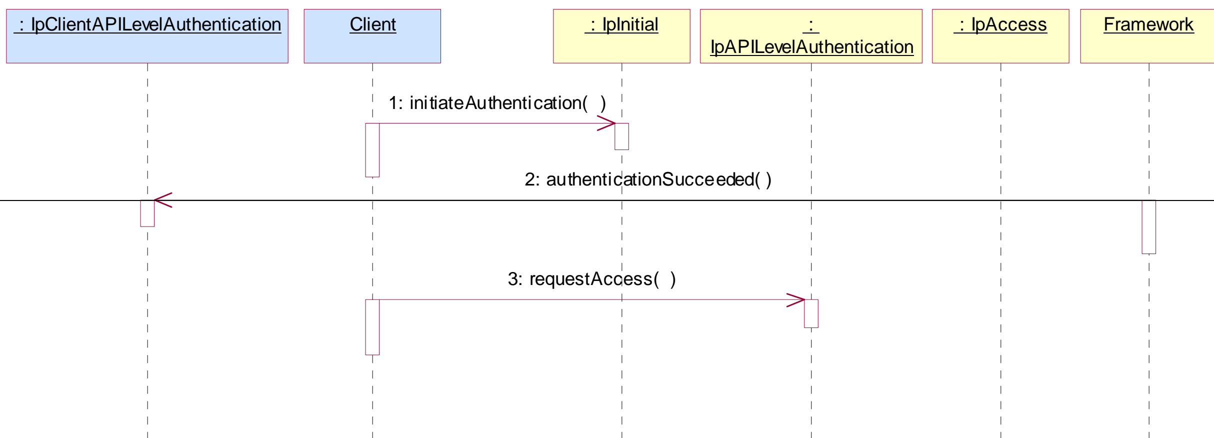
6.1.1 Trust and Security Management Sequence Diagrams

6.1.1.1 Initial Access for trusted parties

The following figure shows a trusted party, typically within the same domain as the Framework, accessing the OSA Framework for the first time. Trusted parties do not need to be authenticated and after contacting the Initial interface the Framework will indicate that no further authentication is needed and that the application can immediately gain access to other framework interfaces and SCFs. This is done by invoking the requestAccess method.


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthenticationWithVersion ()
2: selectAuthenticationMechanism( )
3: authenticationSucceeded( )
4: requestAccess( )


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthentication( )
2: selectAuthenticationMechanism( )
3: authenticationSucceeded( )
4: requestAccess( )


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthentication( )
2: authenticationSucceeded( )
3: requestAccess( )

1: The Client invokes initiateAuthenticationWithVersion on the Framework's "public" (initial contact) interface to initiate the authentication process. It provides in turn a reference to its own authentication interface. The Framework returns a reference to its authentication interface.

~~2: Before authentication can begin, the client shall invoke selectAuthenticationMechanism. Although authentication will not take place this time, failure to invoke this method may prevent the Framework from ever re-authenticating the Client, if it wishes to at a later stage.~~

~~2~~32: Based on the domainID information that was supplied in the Initiate Authentication step, the Framework knows it deals with a trusted party and no further authentication is needed. Therefore the Framework provides the authentication succeeded indication.

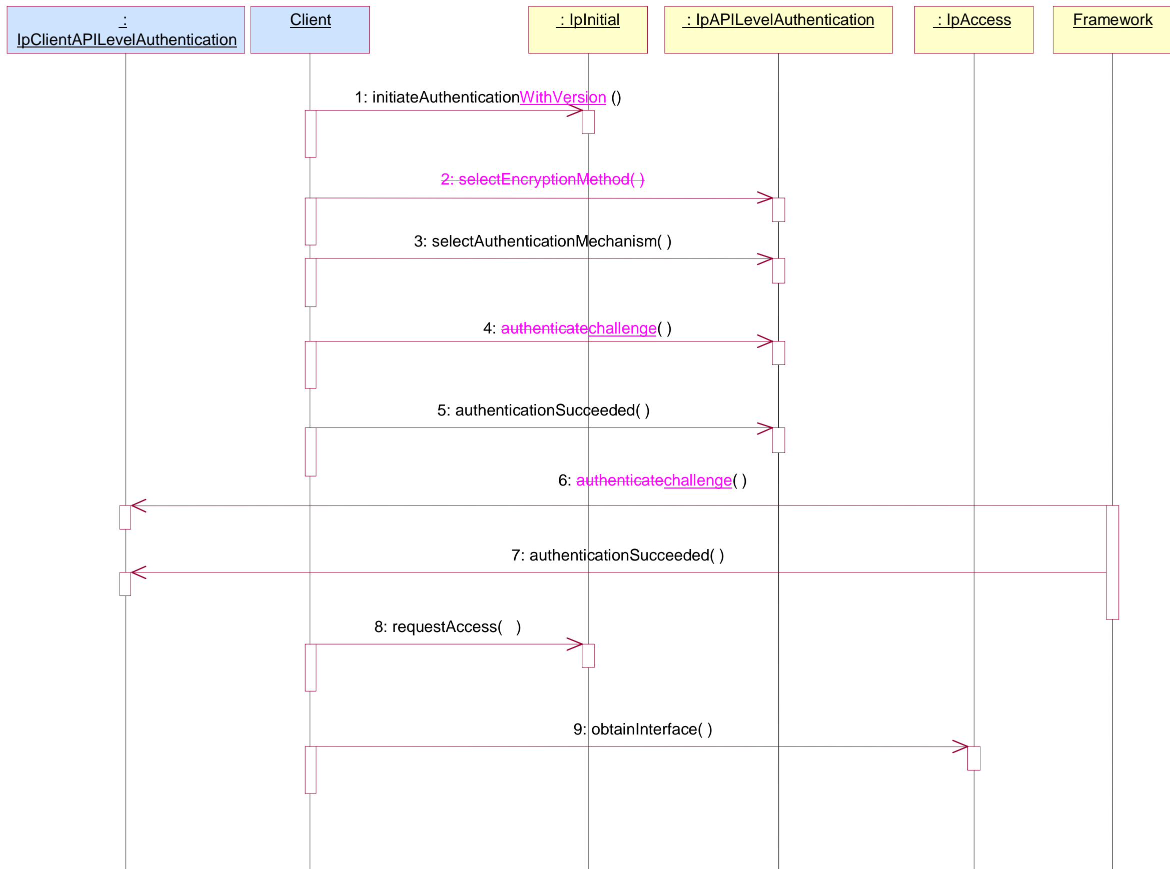
~~3~~43: The Client invokes requestAccess on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface. The Framework returns a reference to its access interface.

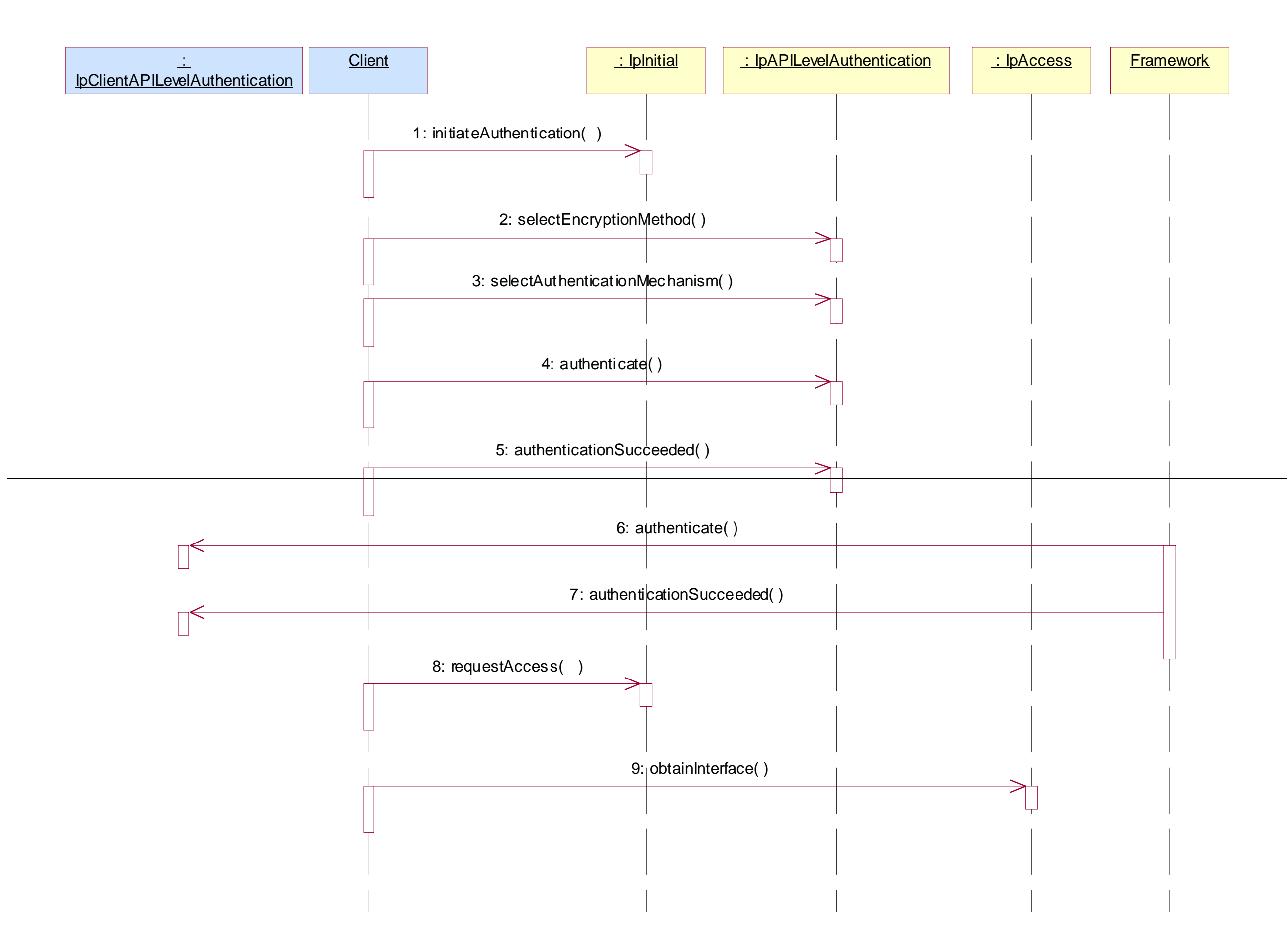
6.1.1.2 Initial Access

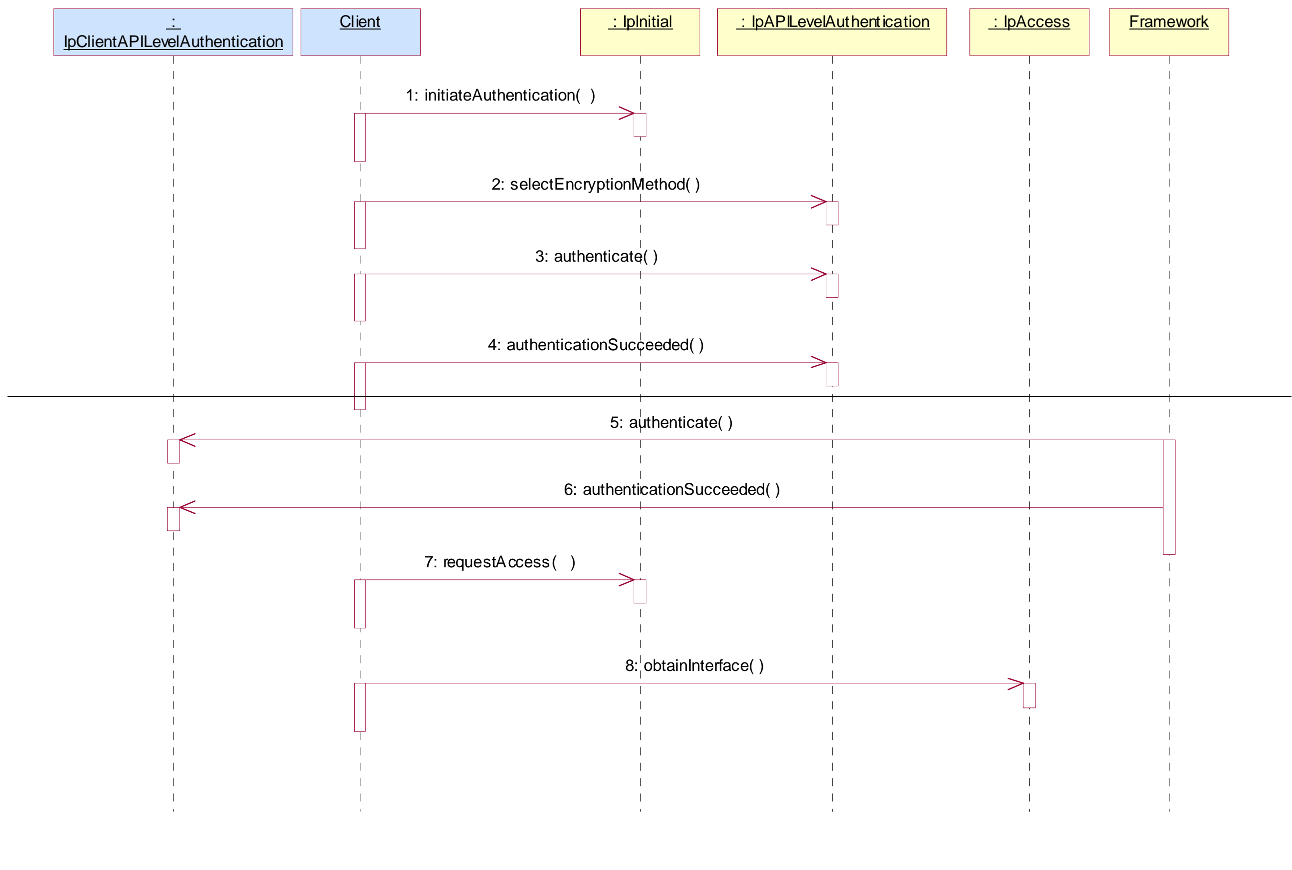
The following figure shows a client accessing the OSA Framework for the first time.

Before being authorized to use the OSA SCFs, the client must first of all authenticate itself with the Framework. For this purpose the client needs a reference to the Initial Contact interfaces for the Framework; this may be obtained through a URL, a Naming or Trading Service or an equivalent service, a stringified object reference, etc. At this stage, the client has no guarantee that this is a Framework interface reference, but it to initiate the authentication process with the Framework. The Initial Contact interface supports only the initiateAuthenticationWithVersion method to allow the authentication process to take place.

Once the client has authenticated with the Framework, it can gain access to other framework interfaces and SCFs. This is done by invoking the requestAccess method, by which the client requests a certain type of access SCF.


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthenticationWithVersion ()
2: selectEncryptionMethod( )
3: selectAuthenticationMechanism( )
4: authenticatechallenge( )
5: authenticationSucceeded( )
6: authenticatechallenge( )
7: authenticationSucceeded( )
8: requestAccess( )
9: obtainInterface( )


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthentication( )
2: selectEncryptionMethod( )
3: selectAuthenticationMechanism( )
4: authenticate( )
5: authenticationSucceeded( )
6: authenticate( )
7: authenticationSucceeded( )
8: requestAccess( )
9: obtainInterface( )


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthentication( )
2: selectEncryptionMethod( )
3: authenticate( )
4: authenticationSucceeded( )
5: authenticate( )
6: authenticationSucceeded( )
7: requestAccess( )
8: obtainInterface( )

1: Initiate Authentication

The client invokes initiateAuthenticationWithVersion on the Framework's "public" (initial contact) interface to initiate the authentication process. It provides in turn a reference to its own authentication interface. The Framework returns a reference to its authentication interface.

~~2: Select Encryption Method~~

~~The client invokes selectEncryptionMethod on the Framework's API Level Authentication interface, identifying the encryption methods it supports. The Framework prescribes the method to be used.~~

3: Select Authentication Mechanism

The client invokes selectAuthenticationMechanism on the Framework's API Level Authentication interface, identifying the authentication algorithm it supports for use with CHAP authentication. The Framework prescribes the method to be used. OSA authentication is based on CHAP, which prescribes the MD5 hashing algorithm as the minimum to be supported. Note however that the framework need not accept this algorithm.

~~3~~4: ~~Authenticate~~Challenge

~~4~~5: The client provides an indication if authentication succeeded. ~~CHAP prescribes the MD5 hashing algorithm as the minimum to be supported. Note however that the client need not accept this algorithm.~~

~~5~~6: The client and Framework authenticate each other. The sequence diagram illustrates one of a series of one or more invocations of the ~~authenticate~~ challenge method on the Framework's API Level Authentication interface. In each invocation, the client supplies a challenge and the Framework returns the correct response. Alternatively or additionally the Framework may issue its own challenges to the client using the authenticate method on the client's API Level Authentication interface.

~~6~~7: The Framework provides an indication if authentication succeeded.

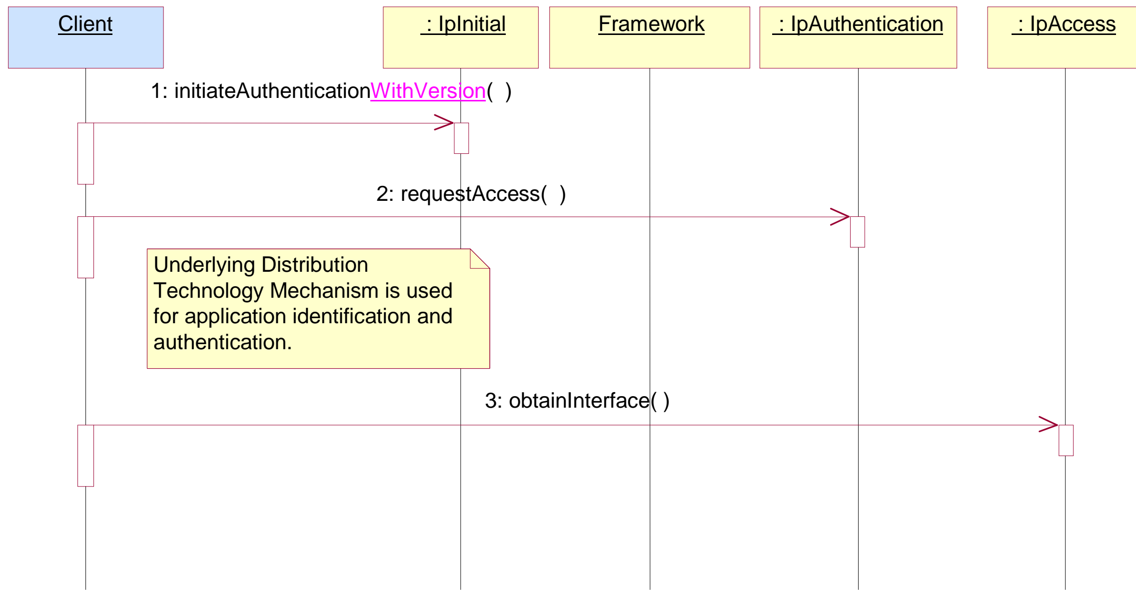
~~7~~8: Request Access

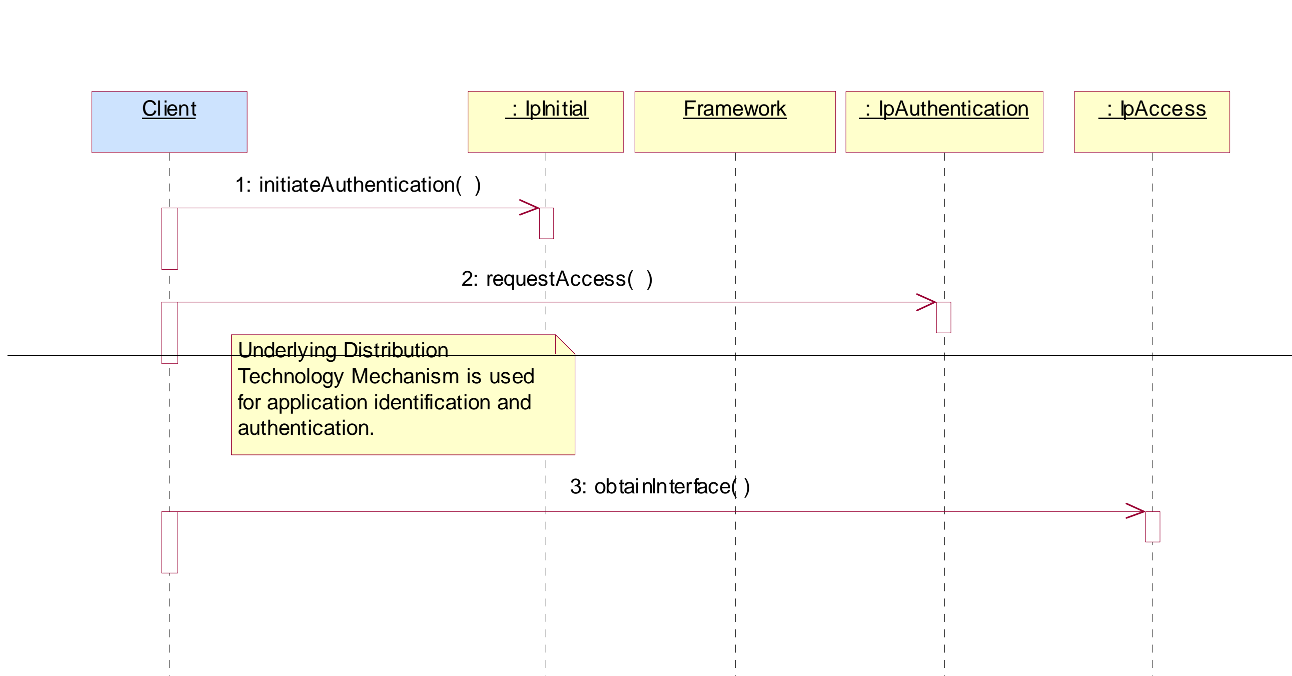
Upon successful (mutual) authentication, the client invokes requestAccess on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface. The Framework returns a reference to its access interface.

~~8~~9: The client invokes obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

6.1.1.3 Authentication

This sequence diagram illustrates the two-way mechanism by which the client and the framework mutually authenticate one another using an underlying distribution technology mechanism.





1: The client calls initiateAuthenticationWithVersion on the OSA Framework Initial interface. This allows the client to specify the type of authentication process. In this case, the client selects to use the underlying distribution technology mechanism for identification and authentication.

2: The client invokes the requestAccess method on the Framework's Authentication interface. The Framework now uses the underlying distribution technology mechanism for identification and authentication of the client.

3: If the authentication was successful, the client can now invoke obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

6.1.1.4 API Level Authentication

This sequence diagram illustrates the two-way mechanism by which the client and the framework mutually authenticate one another.

The OSA API supports multiple authentication techniques. The procedure used to select an appropriate technique for a given situation is described below. The authentication mechanisms may be supported by cryptographic processes to provide confidentiality, and by digital signatures to ensure integrity. The inclusion of cryptographic processes and digital signatures in the authentication procedure depends on the type of authentication technique selected. In some cases strong authentication may need to be enforced by the Framework to prevent misuse of resources. In addition it may be necessary to define the minimum encryption key length that can be used to ensure a high degree of confidentiality.

The client must authenticate with the Framework before it is able to use any of the other interfaces supported by the Framework. Invocations on other interfaces will fail until authentication has been successfully completed.

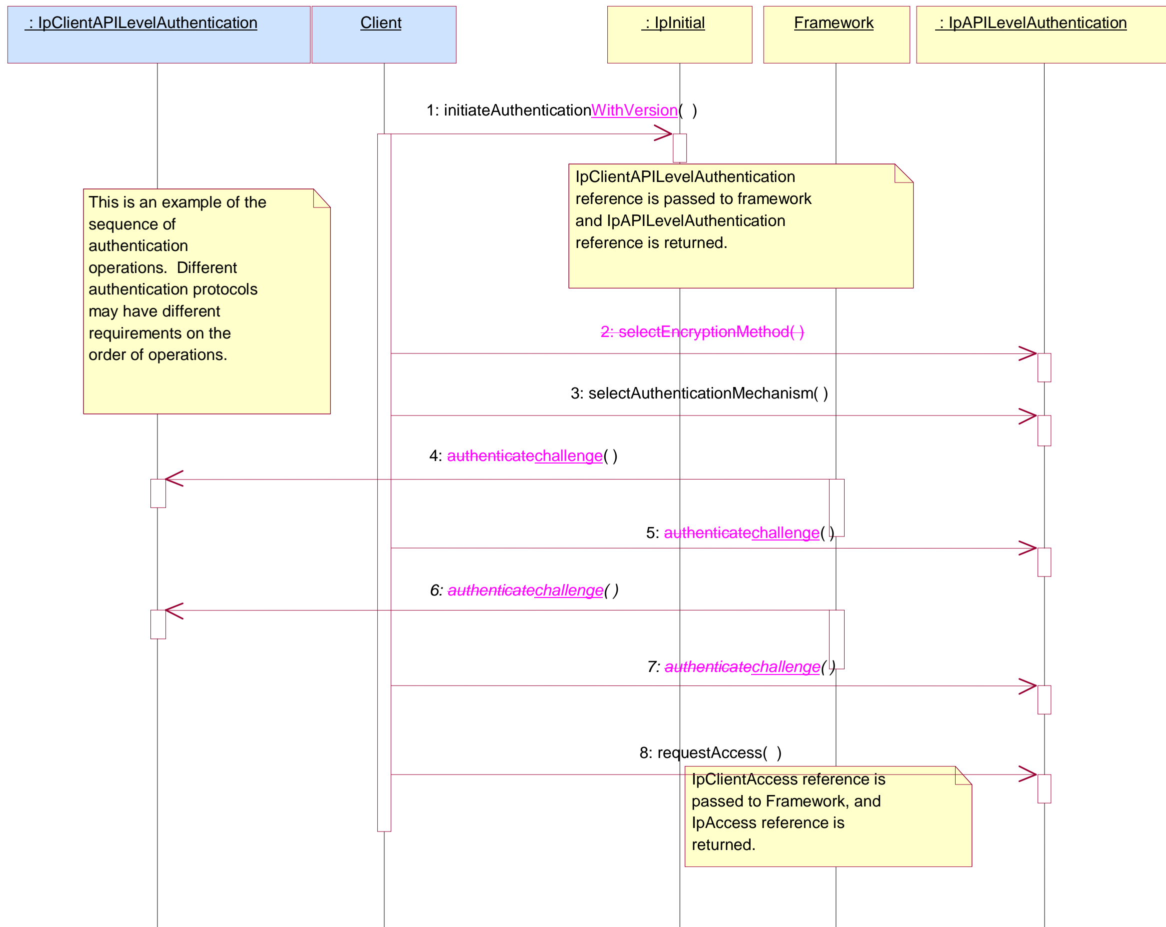
1) The client calls initiateAuthenticationWithVersion on the OSA Framework Initial interface. This allows the client to specify the type of authentication process. This authentication process may be specific to the provider, or the implementation technology used. The initiateAuthenticationWithVersion method can be used to specify the specific process, (e.g. CORBA security). OSA defines a generic authentication interface (API Level Authentication), which can be used to perform the authentication process. The initiateAuthenticationWithVersion method allows the client to pass a reference to its own authentication interface to the Framework, and receive a reference to the authentication interface preferred by the client, in return. In this case the API Level Authentication interface.

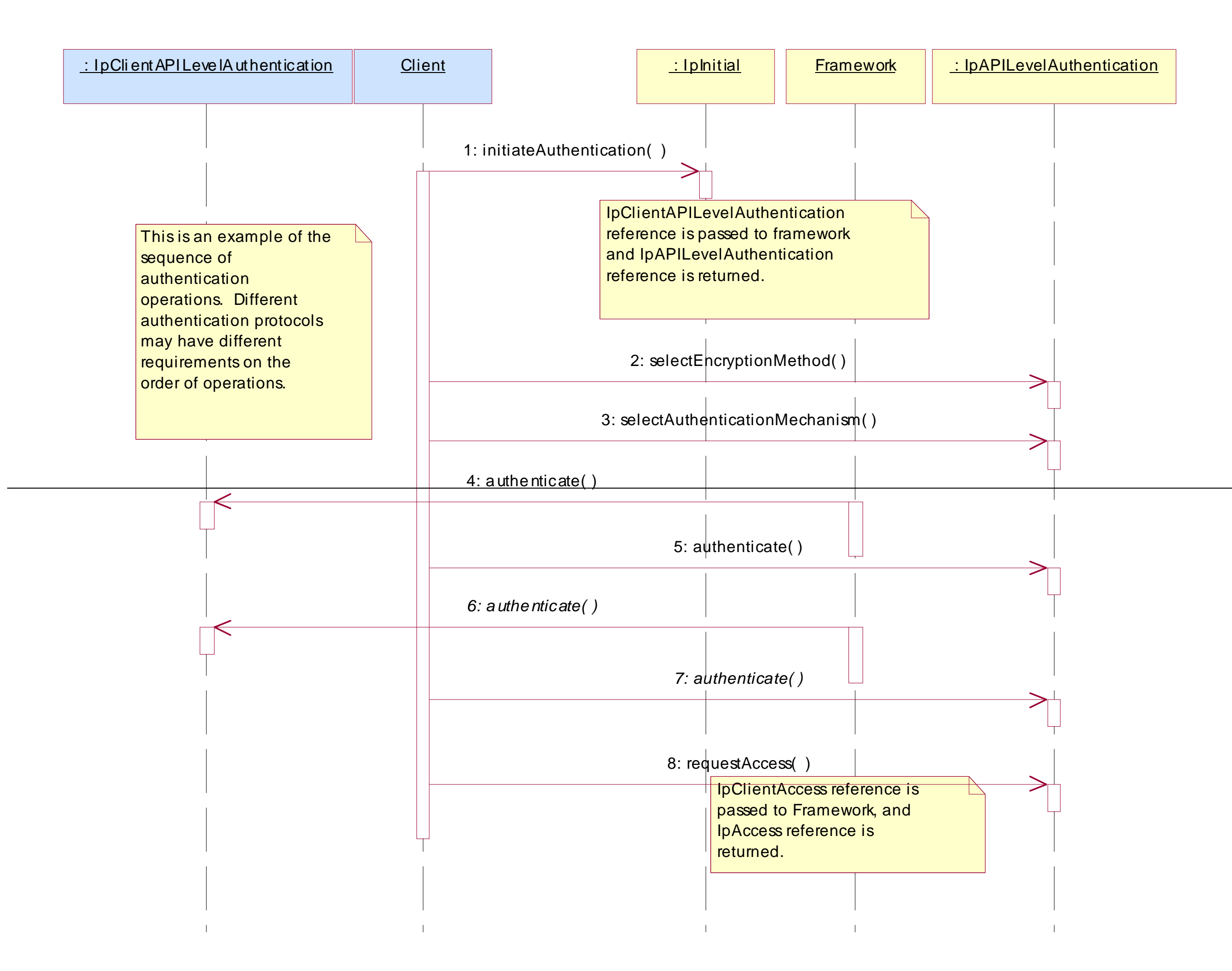
~~2) The client invokes the selectEncryptionMethod on the Framework's API Level Authentication interface. This includes the encryption capabilities of the client. The framework then chooses an encryption method based on the encryption capabilities of the client and the Framework. If the client is capable of handling more than one encryption method, then the Framework chooses one option, defined in the prescribedMethod parameter. In some instances, the encryption capability of the client may not fulfil the demands of the Framework, in which case, the authentication will fail.~~

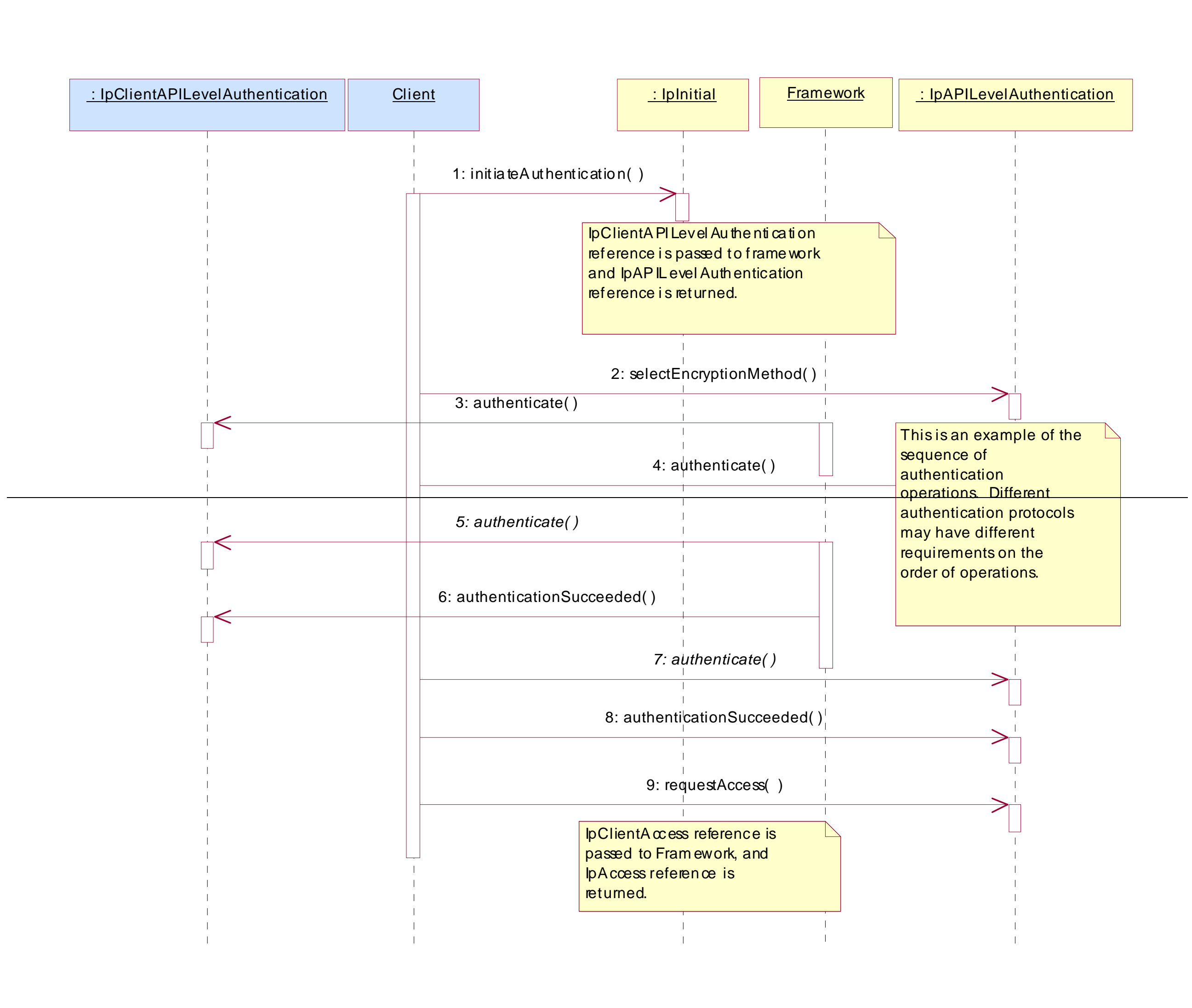
3) The client invokes the selectAuthenticationMechanism on the Framework's API Level Authentication interface. This includes the authentication algorithms supported by the client. The framework then chooses a mechanism based on the capabilities of the client and the Framework. If the client is capable of handling more than one mechanism, then the Framework chooses one option, defined in the prescribedMethod parameter. In some instances, the authentication mechanism of the client may not fulfil the demands of the Framework, in which case, the authentication will fail, for example~~.~~ CHAP prescribes the MD5 hashing algorithm as the minimum to be supported, however the framework need not accept this algorithm.

4~~3~~) The application and Framework interact to authenticate each other by usin~~h~~g the challenge method. For an authentication method of P_OSA_AUTHENTICATION, this procedure consists of a number of challenge/ response exchanges. This authentication protocol is performed using the authenticate method on the API Level Authentication interface. P_OSA_AUTHENTICATION is based on CHAP, which is primarily a one-way protocol. Mutual authentication is achieved by the framework invoking the authenticate method on the client's APILevelAuthentication interface.

Note that at any point during the access session, either side can request re-authentication. Re-authentication does not have to be mutual.


: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentication
1: initiateAuthenticationWithVersion( )
IpClientAPILevelAuthentication reference is passed to framework and IpAPILevelAuthentication reference is returned.
This is an example of the sequence of authentication operations. Different authentication protocols may have different requirements on the order of operations.
2: selectEncryptionMethod( )
3: selectAuthenticationMechanism( )
4: authenticatechallenge( )
5: authenticatechallenge( )
6: authenticatechallenge( )
7: authenticatechallenge( )
8: requestAccess( )
IpClientAccess reference is passed to Framework, and IpAccess reference is returned.


: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentication
1: initiateAuthentication( )
IpClientAPILevelAuthentication reference is passed to framework and IpAPILevelAuthentication reference is returned.
This is an example of the sequence of authentication operations. Different authentication protocols may have different requirements on the order of operations.
2: selectEncryptionMethod( )
3: selectAuthenticationMechanism( )
4: authenticate( )
5: authenticate( )
6: authenticate( )
7: authenticate( )
8: requestAccess( )
IpClientAccess reference is passed to Framework, and IpAccess reference is returned.



: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentication
1: initiateAuthentication( )
IpClientAPILevelAuthentication reference is passed to framework and IpAPILevelAuthentication reference is returned.
2: selectEncryptionMethod( )
3: authenticate( )
This is an example of the sequence of authentication operations. Different authentication protocols may have different requirements on the order of operations.
4: authenticate( )
5: authenticate( )
6: authenticationSucceeded( )
7: authenticate( )
8: authenticationSucceeded( )
9: requestAccess( )
IpClientAccess reference is passed to Framework, and IpAccess reference is returned.

6.1.1.1 Interface Class IpClientAPILevelAuthentication

Inherits from: IpInterface.

<<Interface>>
IpClientAPILevelAuthentication
<<deprecated>> authenticate (challenge : in TpOctetSet) : TpOctetSet abortAuthentication () : void authenticationSucceeded () : void <<new>> challenge (challenge : in TpOctetSet) : TpOctetSet

Method
authenticate()

This method is deprecated and replaced by challenge(). It shall only be used when the deprecated method initiateAuthentication() is used on the IpInitial interface instead of initiateAuthenticationWithVersion). This method will be removed in a later release of the specification.

This method is used by the framework to authenticate the client. The challenge will be encrypted using the mechanism prescribed by selectEncryptionMethod. The client must respond with the correct responses to the challenges presented by the framework. The number of exchanges is dependent on the policies of each side. The whole authentication process is deemed successful when the authenticationSucceeded method is invoked. The invocation of this method may be interleaved with authenticate() calls by the client on the IpAPILevelAuthentication interface.

Returns <response> : This is the response of the client application to the challenge of the framework in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by selectEncryptionMethod().

Parameters

challenge : in TpOctetSet

The challenge presented by the framework to be responded to by the client. The challenge mechanism used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod().

Returns

TpOctetSet

Method
abortAuthentication()

The framework uses this method to abort the authentication process. This method is invoked if the framework wishes to abort the authentication process, (unless the client responded incorrectly to a challenge in which case no further communication with the client should occur.) If this method has been invoked, calls to the requestAccess operation on IpAPILevelAuthentication will return an error code (P_ACCESS_DENIED), until the client has been properly authenticated.

Parameters

No Parameters were identified for this method

Method

authenticationSucceeded()

The Framework uses this method to inform the client of the success of the authentication attempt.

Parameters

No Parameters were identified for this method

Method

challenge()

This method is used by the framework to authenticate the client. The client must respond with the correct responses to the challenges presented by the framework. The number of exchanges is dependent on the policies of each side. The whole authentication process is deemed successful when the authenticationSucceeded method is invoked. The invocation of this method may be interleaved with challenge() calls by the client on the IpAPILevelAuthentication interface.

This method ~~is deprecated and replaced by challenge(). It~~ shall only be used when the ~~deprecated~~ method initiateAuthenticationWithVersion~~initiateAuthentication~~() is used on the IpInitial interface ~~instead of initiateAuthenticationWithVersion)~~.

Returns <response> : This is the response of the client application to the challenge of the framework in the current sequence. The formatting of this parameter shall be according to section 4.1 of RFC 1994. A complete CHAP Response packet shall be used to carry the response string. The Response packet shall make the contents of this returned parameter. The Name field of the CHAP Response packet shall be present but not contain any useful value.

Parameters

challenge : in TpOctetSet

The challenge presented by the framework to be responded to by the client. The challenge format used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996].

~~The challenge shall be formatted in order to include the Initialisation Vector, its length in bytes, and the challenge string, as well as any required padding at the end. The order shall be:~~
~~1 byte indicating the InitialisationVector length~~
~~The Initialisation Vector itself~~
~~The CHAP Request Packet containing the challenge value~~

The formatting of the challenge value shall be according to section 4.1 of RFC 1994. A complete CHAP Request packet shall be used to carry the challenge value. The Name field of the CHAP Request packet shall be present but not contain any useful value.

Returns

TpOctetSet

6.1.1.3 Interface Class IpInitial

Inherits from: IpInterface.

The Initial Framework interface is used by the client to initiate the mutual authentication with the Framework.

<<Interface>> IpInitial
<<deprecated>> initiateAuthentication (clientDomain : in TpAuthDomain, authType : in TpAuthType) : TpAuthDomain <<new>> initiateAuthenticationWithVersion (clientDomain : in TpAuthDomain, authType : in TpAuthType, frameworkVersion : in TpVersion) : TpAuthDomain

6.1.1.3.1 Method <<deprecated>> initiateAuthentication()

This method is deprecated in this version, this means that it will be supported until the next major release of this specification.

This method is invoked by the client to start the process of mutual authentication with the framework, and request the use of a specific authentication method.

Returns <fwDomain> : This provides the client with a framework identifier, and a reference to call the authentication interface of the framework.

```
structure TpAuthDomain {
    domainID:         TpDomainID;
    authInterface:    IpInterfaceRef;
    };
```

The domainID parameter is an identifier for the framework (i.e. TpFwID). It is used to identify the framework to the client.

The authInterface parameter is a reference to the authentication interface of the framework. The type of this interface is defined by the authType parameter. The client uses this interface to authenticate with the framework.

Parameters

clientDomain : in TpAuthDomain

This identifies the client domain to the framework, and provides a reference to the domain's authentication interface.

```
structure TpAuthDomain {
    domainID:         TpDomainID;
    authInterface:    IpInterfaceRef;
};
```

The domainID parameter is an identifier either for a client application (i.e. TpClientAppID) or for an enterprise operator (i.e. TpEntOpID), or for an instance of a registered service (i.e. TpServiceInstanceID) or for a service supplier (i.e. TpServiceSupplierID). It is used to identify the client domain to the framework, (see authenticate() on IpAPILevelAuthentication). If the framework does not recognise the domainID, the framework returns an error code (P_INVALID_DOMAIN_ID).

The authInterface parameter is a reference to call the authentication interface of the client. The type of this interface is defined by the authType parameter. If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

authType : in TpAuthType

This identifies the type of authentication mechanism requested by the client. It provides operators and clients with the opportunity to use an alternative to the API level Authentication interface, e.g. an implementation specific

authentication mechanism like CORBA Security, using the IpAuthentication interface, or Operator specific Authentication interfaces. OSA API level Authentication is the default authentication mechanism (P_OSA_AUTHENTICATION). If P_OSA_AUTHENTICATION is selected, then the clientDomain and fwDomain authInterface parameters are references to interfaces of type Ip(Client)APILevelAuthentication. If P_AUTHENTICATION is selected, the fwDomain authInterface parameter references to interfaces of type IpAuthentication which is used when an underlying distribution technology authentication mechanism is used.

Returns

TpAuthDomain

Raises

TpCommonExceptions, P_INVALID_DOMAIN_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_AUTH_TYPE

6.1.1.3.2 Method <<new>> initiateAuthenticationWithVersion()

This method is invoked by the client to start the process of mutual authentication with the framework, and request the use of a specific authentication method using the new method with support for backward compatibility in the framework. The returned fwDomain authInterface will be selected to match the proposed version from the Client in the Framework response. If the Framework can't work with the proposed framework version the framework returns an error code (P_INVALID_VERSION).

Returns <fwDomain> : This provides the client with a framework identifier, and a reference to call the authentication interface of the framework.

```
structure TpAuthDomain {
    domainID:        TpDomainID;
    authInterface:   IpInterfaceRef;
    };
```

The domainID parameter is an identifier for the framework (i.e. TpFwID). It is used to identify the framework to the client.

The authInterface parameter is a reference to the authentication interface of the framework. The type of this interface is defined by the authType parameter. The client uses this interface to authenticate with the framework.

Parameters

clientDomain : in TpAuthDomain

This identifies the client domain to the framework, and provides a reference to the domain's authentication interface.

```
structure TpAuthDomain {
    domainID:        TpDomainID;
    authInterface:   IpInterfaceRef;
};
```

The domainID parameter is an identifier either for a client application (i.e. TpClientAppID) or for an enterprise operator (i.e. TpEntOpID), or for an instance of a registered service (i.e. TpServiceInstanceID) or for a service supplier (i.e. TpServiceSupplierID). It is used to identify the client domain to the framework, (see ~~authenticate~~challenge() on IpAPILevelAuthentication). If the framework does not recognise the domainID, the framework returns an error code (P_INVALID_DOMAIN_ID).

The authInterface parameter is a reference to call the authentication interface of the client. The type of this interface is defined by the authType parameter. If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

authType : in TpAuthType

This identifies the type of authentication mechanism requested by the client. It provides operators and clients with the opportunity to use an alternative to the API level Authentication interface, e.g. an implementation specific authentication mechanism like CORBA Security, using the IpAuthentication interface, or Operator specific Authentication interfaces. OSA API level Authentication is the default authentication mechanism (P_OSA_AUTHENTICATION). If P_OSA_AUTHENTICATION is selected, then the clientDomain and fwDomain authInterface parameters are references to interfaces of type Ip(Client)APILevelAuthentication. If

P_AUTHENTICATION is selected, the fwDomain authInterface parameter references to interfaces of type IpAuthentication that is used when an underlying distribution technology authentication mechanism is used.

frameworkVersion : in TpVersion

This identifies the version of the Framework implemented in the client. The TpVersion is a String containing the version number. Valid version numbers are defined in the respective framework specification.

Returns

TpAuthDomain

Raises

TpCommonExceptions, P_INVALID_DOMAIN_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_AUTH_TYPE, P_INVALID_VERSION

6.3.1.5 Interface Class IpAPILevelAuthentication

Inherits from: IpAuthentication.

The API Level Authentication Framework interface is used by client to perform its part of the mutual authentication process with the Framework necessary to be allowed to use any of the other interfaces supported by the Framework.

<<Interface>> IpAPILevelAuthentication
<<deprecated>> selectEncryptionMethod (encryptionCaps : in TpEncryptionCapabilityList) : TpEncryptionCapability <<deprecated>> authenticate (challenge : in TpOctetSet) : TpOctetSet abortAuthentication () : void authenticationSucceeded () : void <<new>> selectAuthenticationMechanism (authMechanismList : in TpAuthMechanismList) : TpAuthMechanism <<new>> challenge (challenge : in TpOctetSet) : TpOctetSet

Method
selectEncryptionMethod()

This method is deprecated and replaced by selectAuthenticationMechanism(). It shall only be used when the IpAPILevelAuthentication interface is obtained by using the deprecated method initiateAuthentication() instead of initiateAuthenticationWithVersion() on the IpInitial interface. This method will be removed in a later release.

The client uses this method to initiate the authentication process. The framework returns its preferred mechanism. This should be within capability of the client. If a mechanism that is acceptable to the framework within the capability of the client cannot be found, the framework throws the P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY exception. Once the framework has returned its preferred mechanism, it will wait for a predefined unit of time before invoking the client's authenticate() method (the wait is to ensure that the client can initialise any resources necessary to use the prescribed encryption method).

Returns <prescribedMethod> : This is returned by the framework to indicate the mechanism preferred by the framework for the encryption process. If the value of the prescribedMethod returned by the framework is not understood by the client, it is considered a catastrophic error and the client must abort.

Parameters

encryptionCaps : in TpEncryptionCapabilityList

This is the means by which the encryption mechanisms supported by the client are conveyed to the framework.

Returns

TpEncryptionCapability

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY

Method
authenticate()

This method is deprecated and replaced by challenge(). It shall only be used when the IpAPILevelAuthentication interface is obtained by using the deprecated method initiateAuthentication() instead of initiateAuthenticationWithVersion() on the IpInitial interface~~in combination with selectEncryptionMethod~~. This method will be removed in a later release.

This method is used by the client to authenticate the framework. The challenge will be encrypted using the mechanism prescribed by selectEncryptionMethod. The framework must respond with the correct responses to the challenges presented by the client. The domainID received in the initiateAuthentication() can be used by the framework to reference the correct public key for the client (the key management system is currently outside of the scope of the OSA APIs). The number of exchanges is dependent on the policies of each side. The whole authentication process is deemed successful when the authenticationSucceeded method is invoked. The invocation of this method may be interleaved with authenticate() calls by the framework on the client's APILevelAuthentication interface.

Returns <response> : This is the response of the framework to the challenge of the client in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by selectEncryptionMethod().

Parameters

challenge : in TpOctetSet

The challenge presented by the client to be responded to by the framework. The challenge mechanism used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod().

Returns

TpOctetSet

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method
abortAuthentication()

The client uses this method to abort the authentication process. This method is invoked if the client no longer wishes to continue the authentication process, (unless the client responded incorrectly to a challenge in which case no further communication with the client should occur.) If this method has been invoked, calls to the requestAccess operation on IpAPILevelAuthentication will return an error code (P_ACCESS_DENIED), until the client has been properly authenticated.

Parameters
No Parameters were identified for this method

Raises

TpCommonExceptions,P_ACCESS_DENIED

Method
authenticationSucceeded()

The client uses this method to inform the framework of the success of the authentication attempt.

Parameters
No Parameters were identified for this method

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method
selectAuthenticationMechanism()

The client uses this method to inform the Framework of the different authentication mechanisms it supports as part of API level Authentication. The Framework will select one of the suggested authentication mechanisms and that mechanism shall be used for authentication by both Framework and Client. This method shall be invoked by the client when it receives the interface reference to IpAPILevelAuthentication from the Framework, since until this method is invoked, authentication challenges by the Framework or the client might not be possible. The authentication mechanism chosen as a result of the response to this method remains valid for an instance of IpAPILevelAuthentication and until this method is re-invoked by the client. If a mechanism that is acceptable to the framework within the capability of the client cannot be found, the framework throws the P_NO_ACCEPTABLE_AUTHENTICATION_MECHANISM exception.

This method shall only be used when the IpAPILevelAuthentication interface is obtained by using initiateAuthenticationWithVersion() on the IpInitial interface.

Returns: selectedMechanism. This is the authentication mechanism chosen by the Framework. The chosen mechanism shall be taken from the list of mechanisms proposed by the Client.

Parameters

authMechanismList : in TpAuthMechanismList

The list of authentication mechanisms supported by the client.

Returns

TpAuthMechanism

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_NO_ACCEPTABLE_AUTHENTICATION_MECHANISM

Method
challenge()

This method is used by the client to authenticate the framework. The framework must respond with the correct responses to the challenges presented by the client. ~~The domainID received in the initiateAuthenticationWithVersion() can be used by the framework to reference the correct public key for the client (the key management system is currently~~

outside of the scope of the OSA APIs). The number of exchanges is dependent on the policies of each side. The whole authentication process is deemed successful when the authenticationSucceeded method is invoked. The invocation of this method may be interleaved with challenge() calls by the framework on the client's APILevelAuthentication interface.

This method shall only be used in combination with selectAuthenticationMechanism()when the IpAPILevelAuthentication interface is obtained by using initiateAuthenticationWithVersion() on the IpInitial interface.

Returns <response> : This is the response of the framework to the challenge of the client in the current sequence. The formatting of this parameter shall be according to section 4.1 of RFC 1994. A complete CHAP Response packet shall be used to carry the response string. The Response packet shall make the contents of this returned parameter. The Name field of the CHAP Response packet shall be present but not contain any useful value.

Parameters

challenge : in TpOctetSet

The challenge presented by the client to be responded to by the framework. The challenge format used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996].

The challenge shall be formatted in order to include the Initialisation Vector, its length in bytes, and the challenge string, as well as any required padding at the end. The order shall be:
1 byte indicating the InitialisationVector length
The Initialisation Vector itself
The CHAP Request Packet containing the challenge value

The formatting of the challenge value shall be according to section 4.1 of RFC 1994. A complete CHAP Request packet shall be used to carry the challenge value. The Name field of the CHAP Request packet shall be present but not contain any useful value.

Returns

TpOctetSet

Raises

TpCommonExceptions, P_ACCESS_DENIED

10.3 Trust and Security Management Data Definitions

10.3.1 TpAccessType

This data type is identical to a TpString. This identifies the type of access interface requested by the client application. If they request P_OSA_ACCESS, then a reference to the IpAccess interface is returned. (Network operators can define their own access interfaces to satisfy client requirements for different types of access. These can be selected using the TpAccessType, but should be preceded by the string "SP_". The following value is defined:

String Value	Description
P_OSA_ACCESS	Access using the OSA Access Interfaces: IpAccess and IpClientAccess

10.3.2 TpAuthType

This data type is identical to a TpString. It identifies the type of authentication mechanism requested by the client. It provides Network operators and clients with the opportunity to use an alternative to the OSA API Level Authentication interface. This can for example be an implementation specific authentication mechanism, e.g. CORBA Security, or a proprietary Authentication interface supported by the Network Operator. OSA API Level Authentication is the default authentication method. Other Network operator specific capabilities may also be used, but should be preceded by the string “SP_”. The following values are defined:

String Value	Description
P_OSA_AUTHENTICATION	Authenticate using the OSA API Level Authentication Interfaces: IpAPILevelAuthentication and IpClientAPILevelAuthentication
P_AUTHENTICATION	Authenticate using the implementation specific authentication mechanism, e.g. CORBA Security.

10.3.3 TpEncryptionCapability

This data type is identical to a TpString, and is defined as a string of characters that identify the encryption capabilities that could be supported by the framework. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". Capabilities may be concatenated, using commas (,) as the separation character. The following values are defined.

String Value	Description
NULL	An empty (NULL) string indicates no client capabilities.
P_DES_56	A simple transfer of secret information that is shared between the client application and the Framework with protection against interception on the link provided by the DES algorithm with a 56-bit shared secret key.
P_DES_128	A simple transfer of secret information that is shared between the client entity and the Framework with protection against interception on the link provided by the DES algorithm with a 128-bit shared secret key.
P_RSA_512	A public-key cryptography system providing authentication without prior exchange of secrets using 512-bit keys.
P_RSA_1024	A public-key cryptography system providing authentication without prior exchange of secrets using 1 024-bit keys.

10.3.4 TpEncryptionCapabilityList

This data type is identical to a TpString. It is a string of multiple TpEncryptionCapability concatenated using a comma (,)as the separation character.

10.3.5 TpEndAccessProperties

This data type is of type TpPropertyList. It identifies the actions that the Framework should perform when an application or service capability feature entity ends its access session (e.g. existing service capability or application sessions may be stopped, or left running).

10.3.6 TpAuthDomain

This is `Sequence of Data Elements` containing all the data necessary to identify a domain: the domain identifier, and a reference to the authentication interface of the domain

Sequence Element Name	Sequence Element Type	Description
DomainID	TpDomainID	Identifies the domain for authentication. This identifier is assigned to the domain during the initial contractual agreements, and is valid during the lifetime of the contract.
AuthInterface	IpInterfaceRef	Identifies the authentication interface of the specific entity. This data element has the same lifetime as the domain authentication process, i.e. in principle a new interface reference can be provided each time a domain intends to access another.

10.3.7 TpInterfaceName

This data type is identical to a TpString, and is defined as a string of characters that identify the names of the Framework SCFs that are to be supported by the OSA API. Other Network operator specific SCFs may also be used, but should be preceded by the string "SP_". The following values are defined.

Character String Value	Description
P_DISCOVERY	The name for the Discovery interface.
P_EVENT_NOTIFICATION	The name for the Event Notification interface.
P_OAM	The name for the OA&M interface.
P_LOAD_MANAGER	The name for the Load Manager interface.
P_FAULT_MANAGER	The name for the Fault Manager interface.
P_HEARTBEAT_MANAGEMENT	The name for the Heartbeat Management interface.
P_SERVICE_AGREEMENT_MANAGEMENT	The name of the Service Agreement Management interface.
P_REGISTRATION	The name for the Service Registration interface.
P_ENT_OP_ACCOUNT_MANAGEMENT	The name for the Service Subscription: Enterprise Operator Account Management interface.
P_ENT_OP_ACCOUNT_INFO_QUERY	The name for the Service Subscription: Enterprise Operator Account Information Query interface.
P_SVC_CONTRACT_MANAGEMENT	The name for the Service Subscription: Service Contract Management interface.
P_SVC_CONTRACT_INFO_QUERY	The name for the Service Subscription: Service Contract Information Query interface.
P_CLIENT_APP_MANAGEMENT	The name for the Service Subscription: Client Application Management interface.
P_CLIENT_APP_INFO_QUERY	The name for the Service Subscription: Client Application Information Query interface.
P_SVC_PROFILE_MANAGEMENT	The name for the Service Subscription: Service Profile Management interface.
P_SVC_PROFILE_INFO_QUERY	The name for the Service Subscription: Service Profile Information Query interface.

10.3.8 TpInterfaceNameList

This data type defines a Numbered Set of Data Elements of type TpInterfaceName.

10.3.9 TpServiceToken

This data type is identical to a TpString, and identifies a selected SCF. This is a free format text token returned by the Framework, which can be signed as part of a service agreement. This will contain Network operator specific information relating to the service level agreement. The serviceToken has a limited lifetime, which is the same as the lifetime of the service agreement in normal conditions. If something goes wrong the serviceToken expires, and any method accepting the serviceToken will return an error code (`P_INVALID_SERVICE_TOKEN`). Service Tokens will automatically expire if the client or Framework invokes the endAccess method on the other's corresponding access interface.

10.3.10 TpSignatureAndServiceMgr

This is a Sequence of Data Elements containing the digital signature of the Framework for the service agreement, and a reference to the SCF manager interface of the SCF.

Sequence Element Name	Sequence Element Type
DigitalSignature	TpOctetSet
ServiceMgrInterface	IpServiceRef

The digitalSignature is the signed version of a hash of the service token and agreement text given by the client application.

The ServiceMgrInterface is a reference to the SCF manager interface for the selected SCF.

10.3.11 TpSigningAlgorithm

This data type is identical to a TpString, and is defined as a string of characters that identify the signing algorithm that shall be used. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined.

String Value	Description
NULL	An empty (NULL) string indicates no signing algorithm is required
P_MD5_RSA_512	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public-key cryptography system using a 512-bit key.
P_MD5_RSA_1024	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public- key cryptography system using a 1 024-bit key

10.3.12 TpAuthMechanism

This data type is identical to a TpString. It identifies an authentication mechanism to be used for API Level Authentication. The following values are defined:

String Value	Description
P_OSA_MD5	Authentication is based on the use of MD5 (RFC 1321) hashing algorithm to generate a response based on a shared secret and a challenge received via authenticate() method. The capability to use this algorithm is required to be supported when using CHAP (RFC 1994) but its use is not recommended.
P_OSA_HMAC_SHA1_96	Authentication is based on the use of HMAC-SHA1 (RFC 2404) hashing algorithm to generate a response based on a shared secret and a challenge received via authenticate() method.
P_OSA_HMAC_MD5_96	Authentication is based on the use of HMAC-MD5 (RFC 2403) hashing algorithm to generate a response based on a shared secret and a challenge received via authenticate() method.

10.3.13 TpAuthMechanismList

This data type is identical to a TpString. It is a string of multiple TpAuthMechanism concatenated using a comma (,)as the separation character.

11 Exception Classes

The following are the list of exception classes which are used in this interface of the API.

Name	Description
P_ACCESS_DENIED	The client is not currently authenticated with the framework
P_APPLICATION_NOT_ACTIVATED	An application is unauthorised to access information and request services with regards to users that have deactivated that particular application.
P_DUPLICATE_PROPERTY_NAME	A duplicate property name has been received
P_ILLEGAL_SERVICE_ID	Illegal Service ID
P_ILLEGAL_SERVICE_TYPE	Illegal Service Type
P_INVALID_ACCESS_TYPE	The framework does not support the type of access interface requested by the client.
P_INVALID_ACTIVITY_TEST_ID	ID does not correspond to a valid activity test request
P_INVALID_AGREEMENT_TEXT	Invalid agreement text
P_INVALID_ENCRYPTION_CAPABILITY	Invalid encryption capability
P_INVALID_AUTH_TYPE	Invalid type of authentication mechanism
P_INVALID_CLIENT_APP_ID	Invalid Client Application ID
P_INVALID_DOMAIN_ID	Invalid client ID
P_INVALID_ENT_OP_ID	Invalid Enterprise Operator ID
P_INVALID_PROPERTY	The framework does not recognise the property supplied by the client
P_INVALID_SAG_ID	Invalid Subscription Assignment Group ID
P_INVALID_SERVICE_CONTRACT_ID	Invalid Service Contract ID
P_INVALID_SERVICE_ID	Invalid service ID
P_INVALID_SERVICE_PROFILE_ID	Invalid service profile ID
P_INVALID_SERVICE_TOKEN	The service token has not been issued, or it has expired.
P_INVALID_SERVICE_TYPE	Invalid Service Type
P_INVALID_SIGNATURE	Invalid digital signature
P_INVALID_SIGNING_ALGORITHM	Invalid signing algorithm
P_MISSING_MANDATORY_PROPERTY	Mandatory Property Missing
P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY	~~An~~ No encryption mechanism, which is acceptable to the framework, is ~~not~~ supported by the client
P_NO_ACCEPTABLE_AUTHENTICATION_MECHANISM	No authentication mechanism, which is acceptable to the framework, is supported by the client
P_PROPERTY_TYPE_MISMATCH	Property Type Mismatch
P_SERVICE_ACCESS_DENIED	The client application is not allowed to access this service.
P_SERVICE_NOT_ENABLED	The service ID does not correspond to a service that has been enabled
P_SERVICE_TYPE_UNAVAILABLE	The service type is not available according to the Framework.
P_UNKNOWN_SERVICE_ID	Unknown Service ID
P_UNKNOWN_SERVICE_TYPE	Unknown Service Type

Each exception class contains the following structure:

Structure Element Name	Structure Element Type	Structure Element Description
ExtraInformation	TpString	Carries extra information to help identify the source of the exception, e.g. a parameter name

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020704~~1~~**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03 CR CRNum** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: ⌘ (U)SIM [ ] ME/UE [ ] Radio Access Network [ ] Core Network [X]

Title: ⌘	Correct use of electronic signatures
Source: ⌘	Chelo Abarca, Alcatel Ultan Mulligan, ETSI
Work item code: ⌘	OSA2
Date: ⌘	12/07/2002
Category: ⌘	**F**
Release: ⌘	*REL-5*

Use one of the following categories:
- ***F*** *(correction)*
- ***A*** *(corresponds to a correction in an earlier release)*
- ***B*** *(addition of feature),*
- ***C*** *(functional modification of feature)*
- ***D*** *(editorial modification)*

Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:
- *2 (GSM Phase 2)*
- *R96 (Release 1996)*
- *R97 (Release 1997)*
- *R98 (Release 1998)*
- *R99 (Release 1999)*
- *REL-4 (Release 4)*
- *REL-5 (Release 5)*

Reason for change: ⌘

This CR proposes an overhaul of the digital signature usage in OSA. The changes in this CR are based on those proposed in contributions N5-020283.

Digital signatures are used in OSA for the signing of service agreements. They are also used for the termination of service agreements, and for the Framework's termination of the client's access session. But they are not used for other methods which result in termination of service agreements: those invoked by a client which terminate a client's access session with the Framework. This is a potential security hole, offering a means to perform denial of service attacks.

There is no negotiation mechanism in the API to enable negotiation of the signing algorithms, yet negotiation is used for such things as encryption capabiltities in Authentication.

The choice of signing algorithms is resticted and should be extended with newer choices.

IpClientAccess.terminateAccess() has had correct digital Signature added, including replay protection. Also, functionality extended to close also all service instances associated with access session.

TpSigningAlgorithm extended with state of the art signing algorithms.

IpAccess.endAccess replaced with terminateAccess: to add digital signature for security, to prevent denial of service attacks on this unprotected method. Also to remove the endAccessProperties which were undefined, but without which the method would throw an exception. This removes possiblity to leave service instances open following close of Framework access session, which was a further security hole.

IpAccess.releaseInterface() replaced with relinquishInterface, to add digital signature parameters for security, to prevent denial of service attacks on this

	unprotected method.
Summary of change: ⌘	Add a negotiation mechanism for Signing Algorighms: selectSigningAlgorithm added to provide negotiation for algorithm to be used for ALL digital signatures (even those in signServiceAgreement). TpSigningAlgorithm extended with state of the art signing algorithms. IpClientAccess.terminateAccess() has had correct digital Signature added, including replay protection (timestamp). Also, functionality extended to close also all service instances associated with access session. IpAccess.endAccess replaced with terminateAccess: to add digital signature for security, to prevent denial of service attacks on this unprotected method. Also to remove the endAccessProperties which were undefined, but without which the method would throw an exception. This removes possiblity to leave service instances open following close of Framework access session, which was a further security hole. IpAccess.releaseInterface() replaced with relinquishInterface, to add digital signature parameters for security, to prevent denial of service attacks on this unprotected method. Methods signServiceAgreement() and terminateServiceAgreement clarified to use the signing algorithm negotiated earlier, and to include replay protection using timestamping.
Consequences if not approved: ⌘	Security weaknesses will remain in the OSA specifications, which will limit their adoption and use. Lack of clear instructions for implementors will lead to interoperability difficulties.

Clauses affected: ⌘	6.3, 7.3, 10.3, 11		
Other specs affected: ⌘	☐	Other core specifications ⌘	
	☐	Test specifications	
	☐	O&M Specifications	
Other comments: ⌘			

How to create CRs using this form:

Comprehensive information and tips about how to create CRs can be found at: http://www.3gpp.org/3G_Specs/CRs.htm. Below is a brief summary:

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.

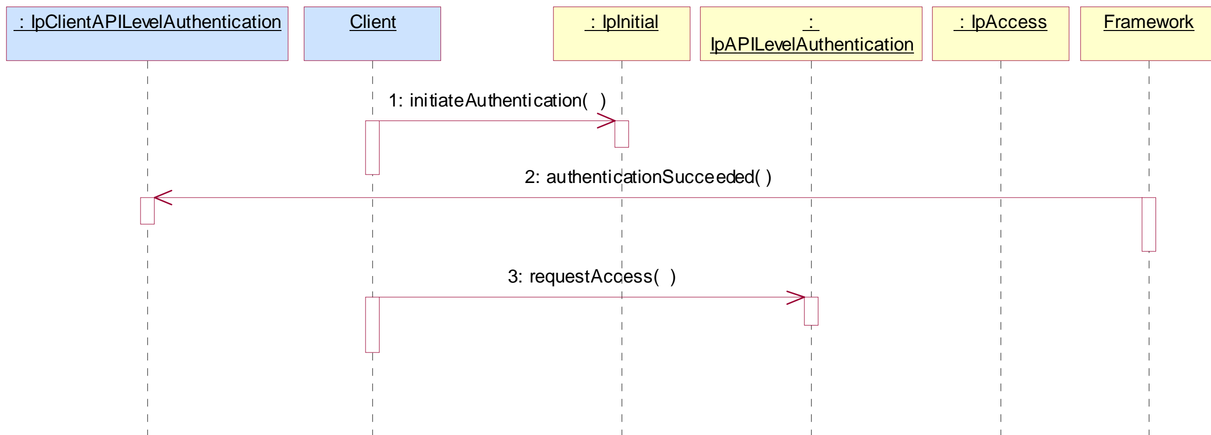
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.

3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

6.1.1 Trust and Security Management Sequence Diagrams

6.1.1.1 Initial Access for trusted parties

The following figure shows a trusted party, typically within the same domain as the Framework, accessing the OSA Framework for the first time. Trusted parties do not need to be authenticated and after contacting the Initial interface the Framework will indicate that no further authentication is needed and that the application can immediately gain access to other framework interfaces and SCFs. This is done by invoking the requestAccess method.



1: The Client invokes initiateAuthentication on the Framework's "public" (initial contact) interface to initiate the authentication process. It provides in turn a reference to its own authentication interface. The Framework returns a reference to its authentication interface.

2: Based on the domainID information that was supplied in the Initiate Authentication step, the Framework knows it deals with a trusted party and no further authentication is needed. Therefore the Framework provides the authentication succeeded indication.

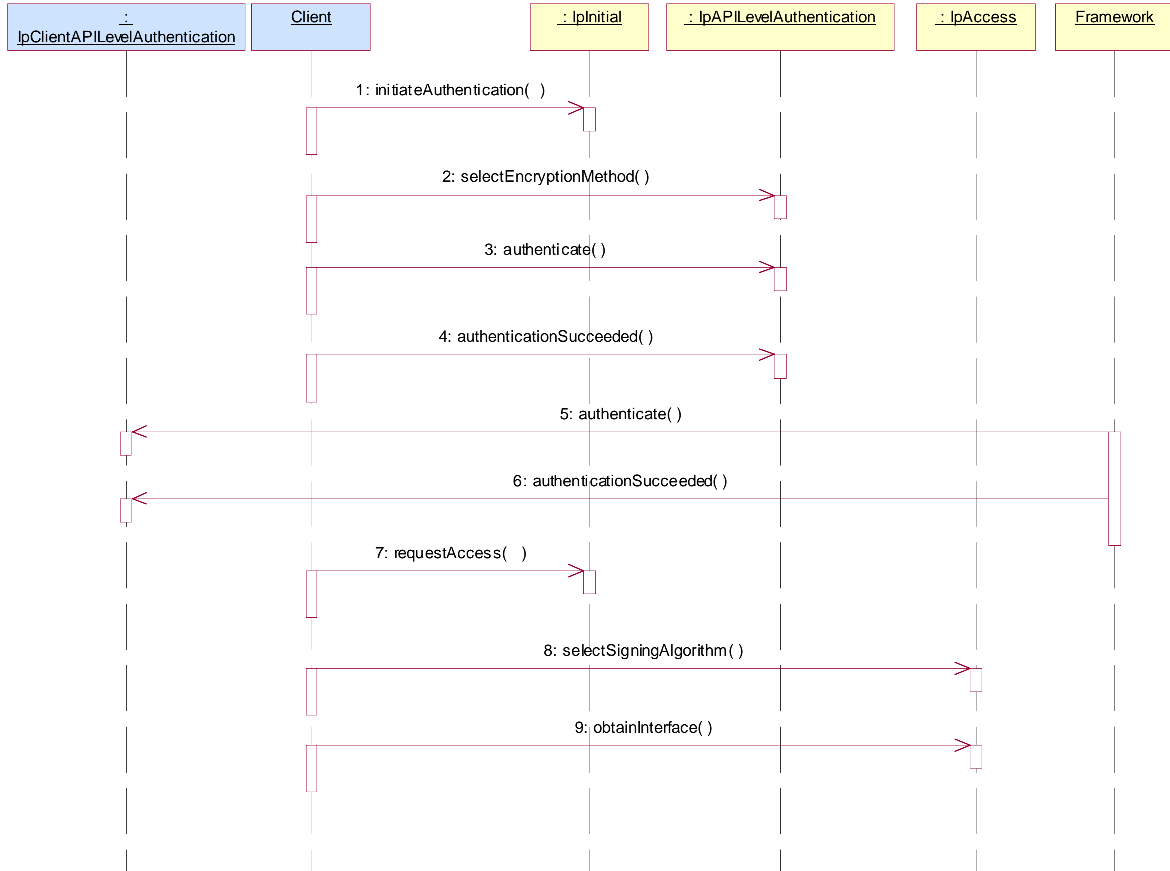
3: The Client invokes requestAccess on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface. The Framework returns a reference to its access interface.

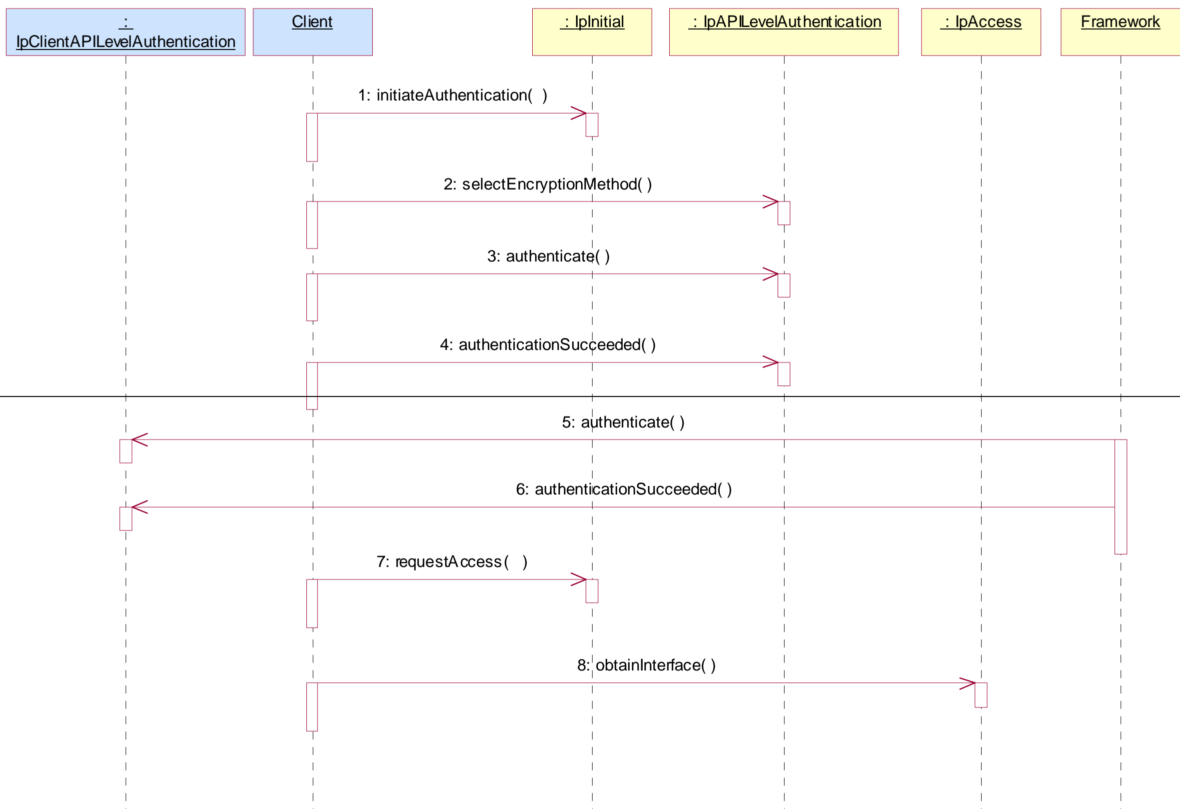
6.1.1.2 Initial Access

The following figure shows a client accessing the OSA Framework for the first time.

Before being authorized to use the OSA SCFs, the client must first of all authenticate itself with the Framework. For this purpose the client needs a reference to the Initial Contact interfaces for the Framework; this may be obtained through a URL, a Naming or Trading Service or an equivalent service, a stringified object reference, etc. At this stage, the client has no guarantee that this is a Framework interface reference, but it to initiate the authentication process with the Framework. The Initial Contact interface supports only the initiateAuthentication method to allow the authentication process to take place.

Once the client has authenticated with the Framework, it can gain access to other framework interfaces and SCFs. This is done by invoking the requestAccess method, by which the client requests a certain type of access SCF.


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthentication( )
2: selectEncryptionMethod( )
3: authenticate( )
4: authenticationSucceeded( )
5: authenticate( )
6: authenticationSucceeded( )
7: requestAccess( )
8: selectSigningAlgorithm( )
9: obtainInterface( )


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthentication( )
2: selectEncryptionMethod( )
3: authenticate( )
4: authenticationSucceeded( )
5: authenticate( )
6: authenticationSucceeded( )
7: requestAccess( )
8: obtainInterface( )

1: Initiate Authentication

The client invokes initiateAuthentication on the Framework's "public" (initial contact) interface to initiate the authentication process. It provides in turn a reference to its own authentication interface. The Framework returns a reference to its authentication interface.

2: Select Encryption Method

The client invokes selectEncryptionMethod on the Framework's API Level Authentication interface, identifying the encryption methods it supports. The Framework prescribes the method to be used.

3: Authenticate

4: The client provides an indication if authentication succeeded.

5: The client and Framework authenticate each other. The sequence diagram illustrates one of a series of one or more invocations of the authenticate method on the Framework's API Level Authentication interface. In each invocation, the client supplies a challenge and the Framework returns the correct response. Alternatively or additionally the Framework may issue its own challenges to the client using the authenticate method on the client's API Level Authentication interface.

6: The Framework provides an indication if authentication succeeded.

7: Request Access

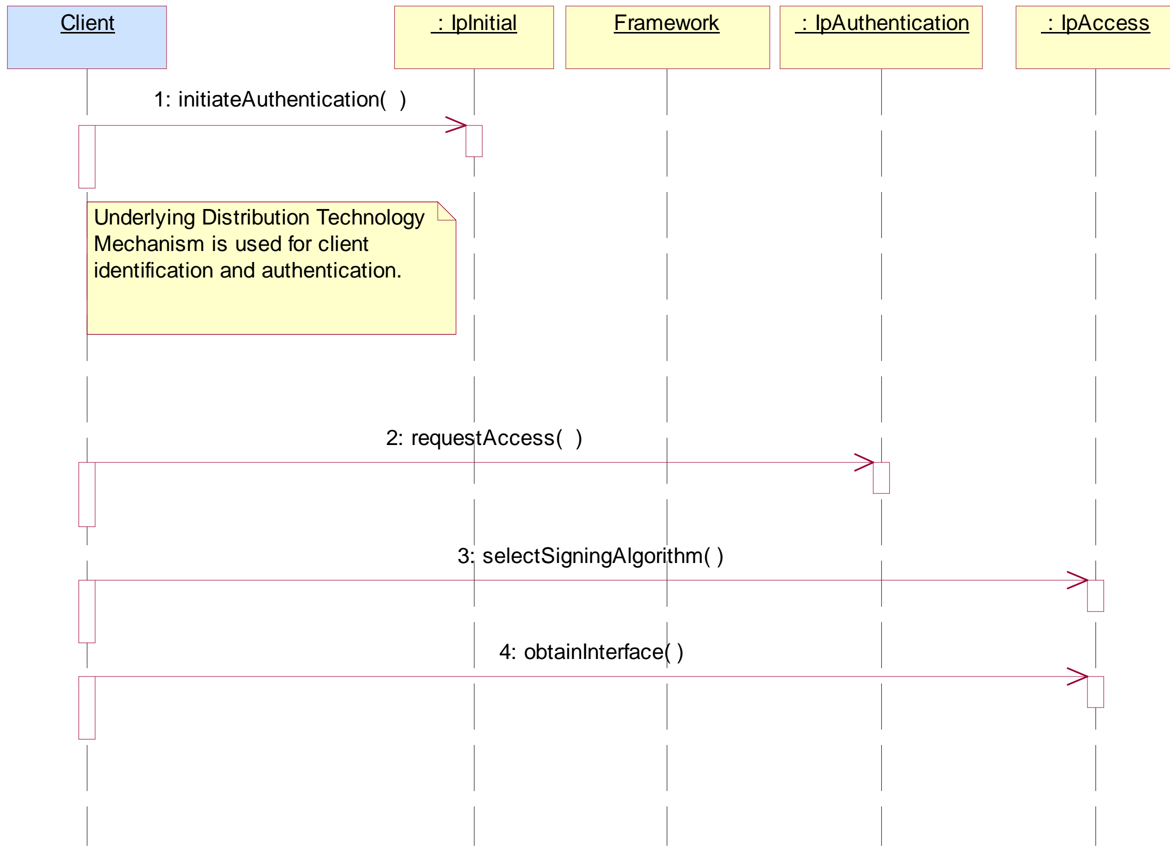
Upon successful (mutual) authentication, the client invokes requestAccess on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface. The Framework returns a reference to its access interface.

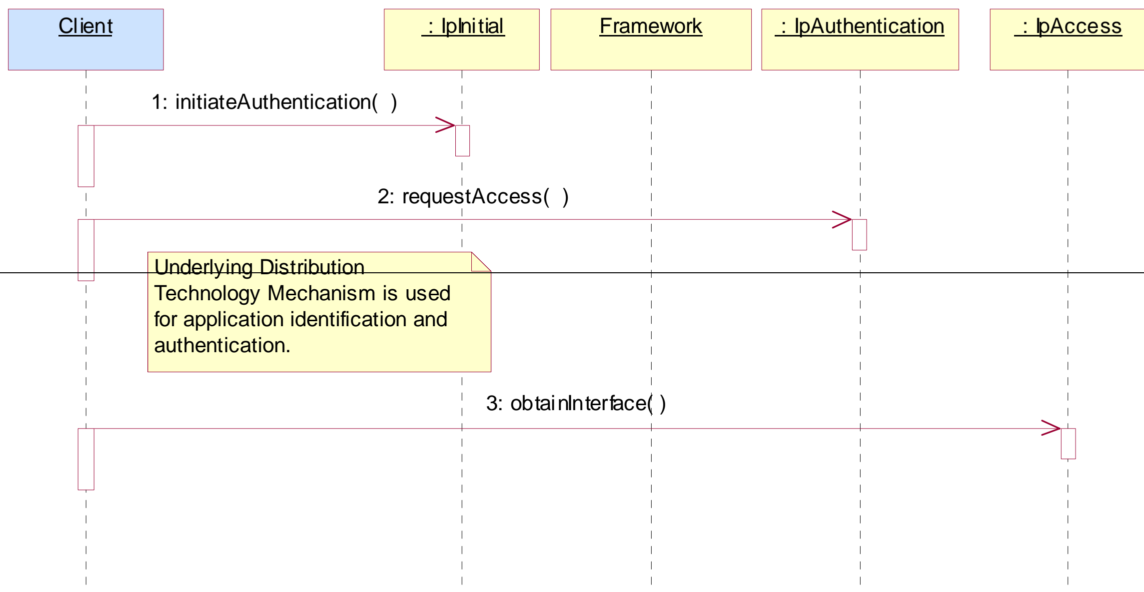
8: The client and framework negotiate the signing algorithm to be used for any signed exchanges.

~~8~~9: The client invokes obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

6.1.1.3 Authentication

This sequence diagram illustrates the two-way mechanism by which the client and the framework mutually authenticate one another using an underlying distribution technology mechanism.





1: The client calls initiateAuthentication on the OSA Framework Initial interface. This allows the client to specify the type of authentication process. In this case, the client selects to use the underlying distribution technology mechanism for identification and authentication.

2: The client invokes the requestAccess method on the Framework's Authentication interface. The Framework now uses the underlying distribution technology mechanism for identification and authentication of the client.

3: If the authentication was successful, the client and the framework can negotiate, on the framework's Access interface, the signing algorithm to be used for any signed exchanges.

34: If the authentication was successful, tThe client can now invoke obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

6.1.1.4 API Level Authentication

This sequence diagram illustrates the two-way mechanism by which the client and the framework mutually authenticate one another.

The OSA API supports multiple authentication techniques. The procedure used to select an appropriate technique for a given situation is described below. The authentication mechanisms may be supported by cryptographic processes to provide confidentiality, and by digital signatures to ensure integrity. The inclusion of cryptographic processes and digital signatures in the authentication procedure depends on the type of authentication technique selected. In some cases strong authentication may need to be enforced by the Framework to prevent misuse of resources. In addition it may be necessary to define the minimum encryption key length that can be used to ensure a high degree of confidentiality.

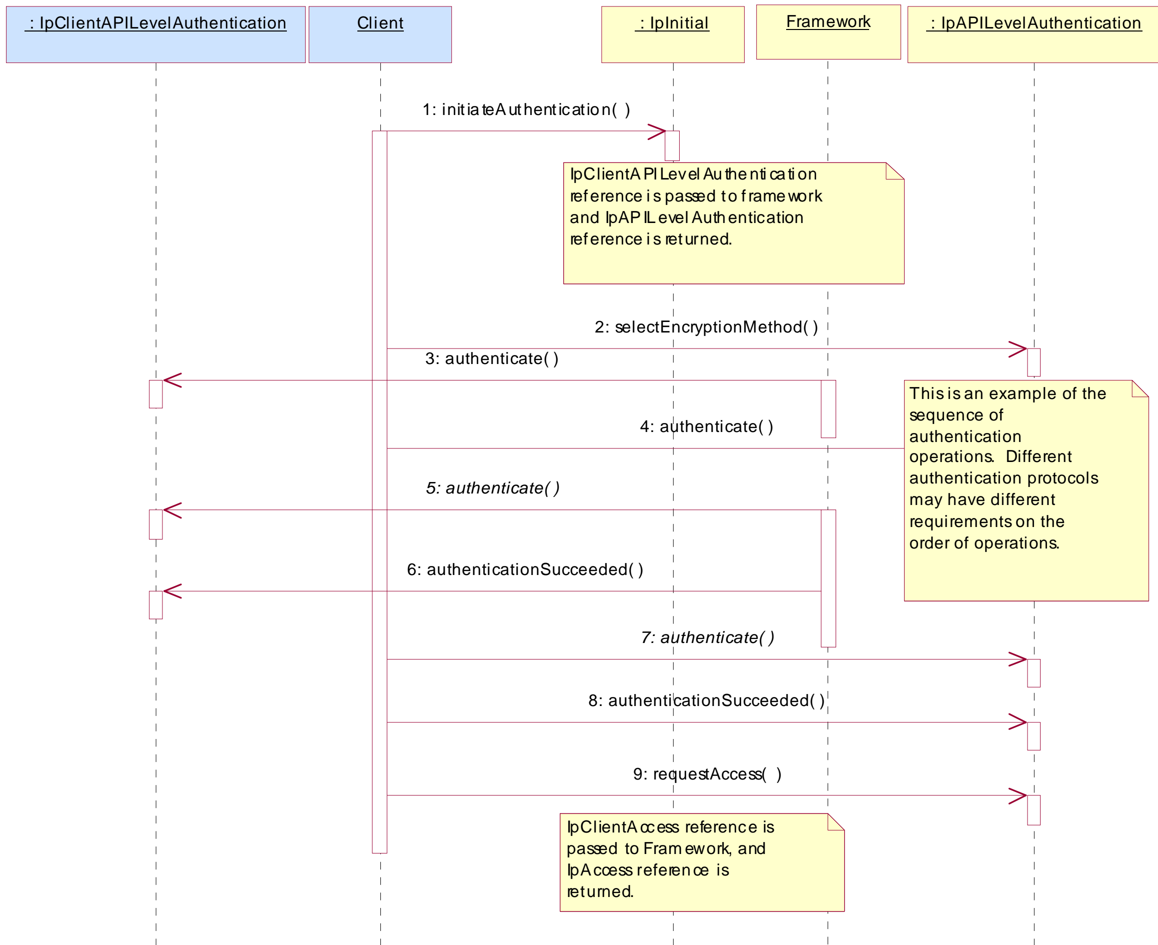
The client must authenticate with the Framework before it is able to use any of the other interfaces supported by the Framework. Invocations on other interfaces will fail until authentication has been successfully completed.

1) The client calls initiateAuthentication on the OSA Framework Initial interface. This allows the client to specify the type of authentication process. This authentication process may be specific to the provider, or the implementation technology used. The initiateAuthentication method can be used to specify the specific process, (e.g. CORBA security). OSA defines a generic authentication interface (API Level Authentication), which can be used to perform the authentication process. The initiateAuthentication method allows the client to pass a reference to its own authentication interface to the Framework, and receive a reference to the authentication interface preferred by the client, in return. In this case the API Level Authentication interface.

2) The client invokes the selectEncryptionMethod on the Framework's API Level Authentication interface. This includes the encryption capabilities of the client. The framework then chooses an encryption method based on the encryption capabilities of the client and the Framework. If the client is capable of handling more than one encryption method, then the Framework chooses one option, defined in the prescribedMethod parameter. In some instances, the encryption capability of the client may not fulfil the demands of the Framework, in which case, the authentication will fail.

3) The application and Framework interact to authenticate each other. For an authentication method of P_OSA_AUTHENTICATION, this procedure consists of a number of challenge/ response exchanges. This authentication protocol is performed using the authenticate method on the API Level Authentication interface. P_OSA_AUTHENTICATION is based on CHAP, which is primarily a one-way protocol. Mutual authentication is achieved by the framework invoking the authenticate method on the client's APILevelAuthentication interface.

Note that at any point during the access session, either side can request re-authentication. Re-authentication does not have to be mutual.


: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentication
1: initiateAuthentication( )
IpClientAPILevelAuthentication reference is passed to framework and IpAPILevelAuthentication reference is returned.
2: selectEncryptionMethod( )
3: authenticate( )
This is an example of the sequence of authentication operations. Different authentication protocols may have different requirements on the order of operations.
4: authenticate( )
5: authenticate( )
6: authenticationSucceeded( )
7: authenticate( )
8: authenticationSucceeded( )
9: requestAccess( )
IpClientAccess reference is passed to Framework, and IpAccess reference is returned.

6.3.1.2 Interface Class IpClientAccess

Inherits from: IpInterface.

IpClientAccess interface is offered by the client to the framework to allow it to initiate interactions during the access session.

<<Interface>> IpClientAccess
terminateAccess (terminationText : in TpString, signingAlgorithm : in TpSigningAlgorithm, digitalSignature : in TpOctetSet) : void

Method
terminateAccess()

The terminateAccess operation is used by the framework to end the client's access session.

After terminateAccess() is invoked, the client will no longer be authenticated with the framework. The client will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail. Also, all remaining service instances created by the framework either directly in this access session or on behalf of the client during this access session shall be terminated. If at any point the framework's level of confidence in the identity of the client becomes too low, perhaps due to re-authentication failing, the framework should terminate all outstanding service agreements for that client, and should take steps to terminate the client's access session WITHOUT invoking terminateAccess() on the client. This follows a generally accepted security model where the framework has decided that it can no longer trust the client and will therefore sever ALL contact with it.

Parameters

terminationText : in TpString

This is the termination text describes the reason for the termination of the access session.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. It shall be identical to the one chosen by the framework in response to IpAccess.selectSigningAlgorithm(). If the signingAlgorithm is not the chosen one, is invalid, or unknown to the client, the P_INVALID_SIGNING_ALGORITHM exception will be thrown. The list of possible algorithms is as specified in the TpSigningAlgorithm table. The identifier used in this parameter must correspond to the digestAlgorithm and signatureAlgorithm fields in the SignerInfo field in the digitalSignature (see below).

digitalSignature : in TpOctetSet

This contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the termination text. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. ~~This is a signed version of a hash of the termination text.~~ The framework uses this to confirm its identity to the client. The client can check that the terminationText has been signed by the framework. If a match is made, the access session is terminated, otherwise the P_INVALID_SIGNATURE exception will be thrown.

Raises

TpCommonExceptions, P_INVALID_SIGNING_ALGORITHM, P_INVALID_SIGNATURE

6.3.1.6 Interface Class IpAccess

Inherits from: IpInterface.

<<Interface>> IpAccess
obtainInterface (interfaceName : in TpInterfaceName) : IpInterfaceRef obtainInterfaceWithCallback (interfaceName : in TpInterfaceName, clientInterface : in IpInterfaceRef) : IpInterfaceRef <<deprecated>> endAccess (endAccessProperties : in TpEndAccessProperties) : void listInterfaces () : TpInterfaceNameList <<deprecated>> releaseInterface (interfaceName : in TpInterfaceName) : void <<new>> selectSigningAlgorithm (signingAlgorithmCaps : in TpSigningAlgorithmCapabilityList) : TpSigningAlgorithm <<new>> terminateAccess (terminationText : in TpString, digitalSignature : in TpOctetSet) : void <<new>> relinquishInterface (interfaceName : in TpInterfaceName, terminationText : in TpString, digitalSignature : in TpOctetSet) : void

Method
obtainInterface()

This method is used to obtain other framework interfaces. The client uses this method to obtain interface references to other framework interfaces. (The obtainInterfaceWithCallback method should be used if the client is required to supply a callback interface to the framework.)

Returns <fwInterface> : This is the reference to the interface requested.

Parameters

interfaceName : in TpInterfaceName

The name of the framework interface to which a reference to the interface is requested. If the interfaceName is invalid, the framework returns an error code (P_INVALID_INTERFACE_NAME).

Returns

IpInterfaceRef

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_INTERFACE_NAME

Method
obtainInterfaceWithCallback()

This method is used to obtain other framework interfaces. The client uses this method to obtain interface references to other framework interfaces, when it is required to supply a callback interface to the framework. (The obtainInterface method should be used when no callback interface needs to be supplied.)

Returns <fwInterface> : This is the reference to the interface requested.

Parameters

interfaceName : in TpInterfaceName

The name of the framework interface to which a reference to the interface is requested. If the interfaceName is invalid, the framework returns an error code (P_INVALID_INTERFACE_NAME).

clientInterface : in IpInterfaceRef

This is the reference to the client interface, which is used for callbacks. If a client interface is not needed, then this method should not be used. (The obtainInterface method should be used when no callback interface needs to be supplied.) If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

Returns

IpInterfaceRef

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME, P_INVALID_INTERFACE_TYPE

Method
<<deprecated>> endAccess()

The endAccess operation is used by the client to request that its access session with the framework is ended. After it is invoked, the client will no longer be authenticated with the framework. The client will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail.

Parameters

endAccessProperties : in TpEndAccessProperties

This is a list of properties that can be used to tell the framework the actions to perform when ending the access session (e.g. existing service sessions may be stopped, or left running). If a property is not recognised by the framework, an error code (P_INVALID_PROPERTY) is returned.

Raises

TpCommonExceptions,P_ACCESS_DENIED, P_INVALID_PROPERTY

Method
listInterfaces()

The client uses this method to obtain the names of all interfaces supported by the framework. It can then obtain the interfaces it wishes to use using either obtainInterface() or obtainInterfaceWithCallback().

Returns <frameworkInterfaces> : The frameworkInterfaces parameter contains a list of interfaces that the framework makes available.

Parameters

No Parameters were identified for this method

Returns

TpInterfaceNameList

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method

<<deprecated>> releaseInterface()

The client uses this method to release a framework interface that was obtained during this access session.

Parameters

interfaceName : in TpInterfaceName

This is the name of the framework interface which is being released. If the interfaceName is invalid, the framework throws the P_INVALID_INTERFACE_NAME exception. If the interface has not been given to the client during this access session, then the P_TASK_REFUSED exception will be thrown.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME

Method

<<new>> selectSigningAlgorithm()

The client uses this method to inform the Framework of the different signing algorithms it supports for use in all cases where digital signatures are required. The Framework will select one of the suggested algorithms. This method shall be the first method invoked by the client on IpAccess. The algorithm chosen as a result of the response to this method remains valid for an instance of IpAccess and until this method is re-invoked by the client. If an algorithm that is acceptable to the framework within the capability of the client cannot be found, the framework throws the P_NO_ACCEPTABLE_SIGNING_ALGORITHM exception.

Returns: selectedAlgorithm. This is the signing algorithm chosen by the Framework. The chosen algorithm shall be taken from the list proposed by the Client.

Parameters

signingAlgorithmCaps : in TpSigningAlgorithmCapabilityList

The list of signing algorithms supported by the client.

Returns

TpSigningAlgorithm

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_NO_ACCEPTABLE_SIGNING_ALGORITHM

Method

<<new>> terminateAccess()

The terminateAccess method is used by the client to request that its access session with the framework is ended. After it is invoked, the client will no longer be authenticated with the framework. The client will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail. Also, all remaining service instances created by the framework either directly in this access session or on behalf of the client during this access session shall be terminated.

Parameters

terminationText : in TpString

This is the termination text describes the reason for the termination of the access session.

digitalSignature : in TpOctetSet

This contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the termination text. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. The client uses this to confirm its identity to the framework. The framework can check that the terminationText has been signed by the client. If a match is made, the access session is terminated, otherwise the P_INVALID_SIGNATURE exception will be thrown.

Raises

TpCommonExceptions, P_INVALID_SIGNATURE

Method

<<new>> relinquishInterface()

The client uses this method to release an instance of a framework interface that was obtained during this access session.

Parameters

interfaceName : in TpInterfaceName

This is the name of the framework interface which is being released. If the interfaceName is invalid, the framework throws the P_INVALID_INTERFACE_NAME exception. If the interface has not been given to the client during this access session, then the P_TASK_REFUSED exception will be thrown.

terminationText : in TpString

This is the termination text describes the reason for the release of the interface. This text is required simply because the digitalSignature parameter requires a terminationText to sign.

digitalSignature : in TpOctetSet

This contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the termination text. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. The client uses this to confirm its identity to the framework. The framework can check that the terminationText has been signed by the client. If a match is made, the interface is released, otherwise the P_INVALID_SIGNATURE exception will be thrown.

Raises

TpCommonExceptions, P_INVALID_SIGNATURE, P_INVALID_INTERFACE_NAME

7.3.3 Service Agreement Management Interface Classes

7.3.3.1 Interface Class IpAppServiceAgreementManagement

Inherits from: IpInterface.

<<Interface>> IpAppServiceAgreementManagement
signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm) : TpOctetSet terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in TpOctetSet) : void

7.3.3.1.1 Method signServiceAgreement()

Upon receipt of the initiateSignServiceAgrement() method from the client application, this method is used by the framework to request that the client application sign an agreement on the service. The framework provides the service agreement text for the client application to sign. The service manager returned will be configured as per the service level agreement. If the framework uses service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties. If the client application agrees, it signs the service agreement, returning its digital signature to the framework.

Returns <digitalSignature> : This contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the service token and agreement text given by the framework. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the service token and agreement text). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. ~~The digitalSignature is the signed version of a hash of the service token and agreement text given by the framework.~~ If the signature is incorrect the serviceToken will be expired immediately.

Parameters

serviceToken : in TpServiceToken

This is the token returned by the framework in a call to the selectService() method. This token is used to identify the service instance to which this service agreement corresponds. (If the client application selects many services, it can determine which selected service corresponds to the service agreement by matching the service token.) If the serviceToken is invalid, or not known by the client application, then the P_INVALID_SERVICE_TOKEN exception is thrown.

agreementText : in TpString

This is the agreement text that is to be signed by the client application using the private key of the client application. If the agreementText is invalid, then the P_INVALID_AGREEMENT_TEXT exception is thrown.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. It shall be identical to the one chosen by the framework in response to IpAccess.selectSigningAlgorithm(). If the signingAlgorithm is not the chosen one, is invalid, or unknown to the client application, the P_INVALID_SIGNING_ALGORITHM exception is thrown. The list of possible

algorithms is as specified in the TpSigningAlgorithm table. The identifier used in this parameter must correspond to the digestAlgorithm and signatureAlgorithm fields in the SignerInfo field in the digitalSignature (see below).

Returns

TpOctetSet

Raises

TpCommonExceptions, P_INVALID_AGREEMENT_TEXT, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNING_ALGORITHM

7.3.3.1.2 Method terminateServiceAgreement()

This method is used by the framework to terminate an agreement for the service.

Parameters

serviceToken : in TpServiceToken

This is the token passed back from the framework in a previous selectService() method call. This token is used to identify the service agreement to be terminated. If the serviceToken is invalid, or unknown to the client application, the P_INVALID_SERVICE_TOKEN exception will be thrown.

terminationText : in TpString

This is the termination text that describes the reason for the termination of the service agreement.

digitalSignature : in TpOctetSet

This contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the service token and the termination text. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the service token and the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. ~~This is a signed version of a hash of the service token and the termination text.~~ The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement(). The framework uses this to confirm its identity to the client application. The client application can check that the terminationText has been signed by the framework. If a match is made, the service agreement is terminated, otherwise the P_INVALID_SIGNATURE exception will be thrown.

Raises

TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE

7.3.3.2 Interface Class IpServiceAgreementManagement

Inherits from: IpInterface.

<<Interface>> IpServiceAgreementManagement
signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm) : TpSignatureAndServiceMgr terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in TpOctetSet) : void selectService (serviceID : in TpServiceID) : TpServiceToken initiateSignServiceAgreement (serviceToken : in TpServiceToken) : void

7.3.3.2.1 Method signServiceAgreement()

This method is used by the client application to request that the framework sign an agreement on the service, which allows the client application to use the service. If the framework agrees, both parties sign the service agreement, and a reference to the service manager interface of the service is returned to the client application. The service manager returned will be configured as per the service level agreement. If the framework uses service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties. If the client application is not allowed to access the service, then an error code (P_SERVICE_ACCESS_DENIED) is returned.

Returns <signatureAndServiceMgr> : This contains the digital signature of the framework for the service agreement, and a reference to the service manager interface of the service.

```
structure TpSignatureAndServiceMgr {
        digitalSignature:  TpOctetSet;
            serviceMgrInterface:  IpServiceRef;
        };
```

The digitalSignature contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the service token and agreement text given by the client application. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the service token and agreement text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. ~~is the signed version of a hash of the service token and agreement text given by the client application.~~

The serviceMgrInterface is a reference to the service manager interface for the selected service.

Parameters

serviceToken : in TpServiceToken

This is the token returned by the framework in a call to the selectService() method. This token is used to identify the service instance requested by the client application. If the serviceToken is invalid, or has expired, an error code (P_INVALID_SERVICE_TOKEN) is returned.

agreementText : in TpString

This is the agreement text that is to be signed by the framework using the private key of the framework. If the agreementText is invalid, then an error code (P_INVALID_AGREEMENT_TEXT) is returned.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. It shall be identical to the one chosen by the framework in response to IpAccess.selectSigningAlgorithm(). If the signingAlgorithm is not the chosen one, is invalid, or unknown to the framework, an error code (P_INVALID_SIGNING_ALGORITHM) is returned. The list of possible algorithms is as specified in the TpSigningAlgorithm table. The identifier used in this parameter must correspond to the digestAlgorithm and signatureAlgorithm fields in the SignerInfo field in the digitalSignature (see below).

Returns

`TpSignatureAndServiceMgr`

Raises

`TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_AGREEMENT_TEXT,P_INVALID_SERVICE_TOKEN,P_INVALID_SIGNING_ALGORITHM,P_SERVICE_ACCESS_DENIED`

7.3.3.2.2 Method terminateServiceAgreement()

This method is used by the client application to terminate an agreement for the service.

Parameters

`serviceToken : in TpServiceToken`

This is the token passed back from the framework in a previous selectService() method call. This token is used to identify the service agreement to be terminated. If the serviceToken is invalid, or has expired, an error code (P_INVALID_SERVICE_TOKEN) is returned.

`terminationText : in TpString`

This is the termination text that describes the reason for the termination of the service agreement.

`digitalSignature : in TpOctetSet`

This contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the service token and the termination text. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the service token and the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. ~~This is a signed version of a hash of the service token and the termination text.~~ The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement().The framework uses this to check that the terminationText has been signed by the client application. If a match is made, the service agreement is terminated, otherwise an error code (P_INVALID_SIGNATURE) is returned.

Raises

`TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE`

7.3.3.2.3 Method selectService()

This method is used by the client application to identify the service that the client application wishes to use. If the client application is not allowed to access the service, then the P_SERVICE_ACCESS_DENIED exception is thrown. The P_SERVICE_ACCESS_DENIED exception is also thrown if the client attempts to select a service for which it has already signed a service agreement for, and therefore obtained an instance of. This is because there must be only one service instance per client application.

Returns <serviceToken> : This is a free format text token returned by the framework, which can be signed as part of a service agreement. This will contain operator specific information relating to the service level agreement. The serviceToken has a limited lifetime. If the lifetime of the serviceToken expires, a method accepting the serviceToken will return an error code (P_INVALID_SERVICE_TOKEN). Service Tokens will automatically expire if the client application or framework invokes the endAccess method on the other's corresponding access interface.

Parameters

serviceID : in TpServiceID

This identifies the service required. If the serviceID is not recognised by the framework, an error code (P_INVALID_SERVICE_ID) is returned.

Returns

TpServiceToken

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_ID, P_SERVICE_ACCESS_DENIED

7.3.3.2.4 Method initiateSignServiceAgreement()

This method is used by the client application to initiate the sign service agreement process. If the client application is not allowed to initiate the sign service agreement process, the exception (P_SERVICE_ACCESS_DENIED) is thrown.

Parameters

serviceToken : in TpServiceToken

This is the token returned by the framework in a call to the selectService() method. This token is used to identify the service instance requested by the client application. If the serviceToken is invalid, or has expired, the exception (P_INVALID_SERVICE_TOKEN) is thrown.

Raises

TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_SERVICE_ACCESS_DENIED

10.3.11 TpSigningAlgorithm

This data type is identical to a TpString, and is defined as a string of characters that identify the signing algorithm that shall be used. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined.

String Value	Description
NULL	An empty (NULL) string indicates no signing algorithm is required
P_MD5_RSA_512	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public-key cryptography system using a 512-bit modulus. The signature generation follows the process and format defined in RFC 2313 (PKCS#1 v1.5). The use of this signing method is deprecated.MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public-key cryptography system using a 512-bit key.
P_MD5_RSA_1024	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public- key cryptography system using a 1024-bit modulus. .The signature generation follows the process and format defined in RFC 2313 (PKCS#1 v1.5). The use of this signing method is deprecated.MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public- key cryptography system using a 1 024-bit key
P_RSASSA_PKCS1-v1_5_SHA1_1024P_RSASSA_PKCS1_v1_5_SHA1_1024	SHA-1 is used to produce a 160-bit message digest based on the input message to be signed. RSA is then used to generate the signature value, following the process defined in section 8 of RFC 2437 and format defined in section 9.2.1 of RFC 2437. The RSA private/public key pair is using a 1024-bit modulus.
P_SHA1_DSA	SHA-1 is used to produce a 160-bit message digest based on the input message to be signed. DSA is then used to generate the signature value. The signature generation follows the process and format defined in section 7.2.2 of RFC 2459.

10.3.12 TpSigningAlgorithmList

This data type is identical to a TpString. It is a string of multiple TpSigningAlgorithm concatenated using a comma (,)as the separation character.

11 Exception Classes

The following are the list of exception classes which are used in this interface of the API.

Name	Description
P_ACCESS_DENIED	The client is not currently authenticated with the framework
P_APPLICATION_NOT_ACTIVATED	An application is unauthorised to access information and request services with regards to users that have deactivated that particular application.
P_DUPLICATE_PROPERTY_NAME	A duplicate property name has been received
P_ILLEGAL_SERVICE_ID	Illegal Service ID
P_ILLEGAL_SERVICE_TYPE	Illegal Service Type
P_INVALID_ACCESS_TYPE	The framework does not support the type of access interface requested by the client.
P_INVALID_ACTIVITY_TEST_ID	ID does not correspond to a valid activity test request
P_INVALID_AGREEMENT_TEXT	Invalid agreement text
P_INVALID_ENCRYPTION_CAPABILITY	Invalid encryption capability
P_INVALID_AUTH_TYPE	Invalid type of authentication mechanism
P_INVALID_CLIENT_APP_ID	Invalid Client Application ID
P_INVALID_DOMAIN_ID	Invalid client ID
P_INVALID_ENT_OP_ID	Invalid Enterprise Operator ID
P_INVALID_PROPERTY	The framework does not recognise the property supplied by the client
P_INVALID_SAG_ID	Invalid Subscription Assignment Group ID
P_INVALID_SERVICE_CONTRACT_ID	Invalid Service Contract ID
P_INVALID_SERVICE_ID	Invalid service ID
P_INVALID_SERVICE_PROFILE_ID	Invalid service profile ID
P_INVALID_SERVICE_TOKEN	The service token has not been issued, or it has expired.
P_INVALID_SERVICE_TYPE	Invalid Service Type
P_INVALID_SIGNATURE	Invalid digital signature
P_INVALID_SIGNING_ALGORITHM	Invalid signing algorithm
P_MISSING_MANDATORY_PROPERTY	Mandatory Property Missing
P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY	No encryption mechanism, which is acceptable to the framework, is supported by the client
P_NO_ACCEPTABLE_SIGNING_ALGORITHM	No signing algorithm, which is acceptable to the framework, is supported by the client
P_PROPERTY_TYPE_MISMATCH	Property Type Mismatch
P_SERVICE_ACCESS_DENIED	The client application is not allowed to access this service.
P_SERVICE_NOT_ENABLED	The service ID does not correspond to a service that has been enabled
P_SERVICE_TYPE_UNAVAILABLE	The service type is not available according to the Framework.
P_UNKNOWN_SERVICE_ID	Unknown Service ID
P_UNKNOWN_SERVICE_TYPE	Unknown Service Type

Each exception class contains the following structure:

Structure Element Name	Structure Element Type	Structure Element Description
ExtraInformation	TpString	Carries extra information to help identify the source of the exception, e.g. a parameter name