3GPP TSG SA WG3 Security — S3#23 **S3-020246**

May 14-17, 2002

Victoria, Canada

Source: **Alcatel**

Title: **Status on OSA Security**

Document for: **Discussion**

Agenda item: **6.15**

1 Introduction

This document presents the current status with regards to the ongoing discussions on OSA security, following the contributions presented to SA3 at S3#22 meeting. Please note that this is in no way an official report of CN5.

Attached to this contribution are four documents that were submitted to the CN5 meeting in April, which were updates of (S3-020101, S3-020102, S3-020103, S3-020104) based on the discussions held during the S3#22 meeting.

In section 2, we provide status information (as known today) on the various issues raised in each contribution (attached).

2 Status

2.1 Authentication scheme negotiation in OSA

2.1.1 Mechanism for negotiation of authentication scheme

Proposed solution 1 in contribution is the preferred one (use of a new additional method selectAuthenticationMethod() to negotiate the auth scheme).

2.1.2 Negotiation of the signing algorithm used in terminateAccess()

The proposed solution was not accepted because the terminateAccess() function is located on a different Framework interface than the (newly defined) selectAuthenticationMethod() function. Current proposals under discussion are to add a new parameter in requesAccess() or to define a new function selectSigningAlgorithm() in the IpAccess interface. Some preference seems to go in favor of the latter.

2.2 Encryption of challenge in CHAP-based OSA authentication

2.2.1 The need for encrypting the challenge

It was agreed that there is no need for it. However, for backwards compatibility reasons, keeping it is still favoured as it does not harm.

2.2.2 No formatting defined for challenge encryption

Some discussion went on on the need for specifying the padding. Fear was that this would force the application developer to be aware of the padding method used while crypto libraries in use today hide this from the programmer. It is however necessary to

specify the padding scheme together with each encryption algorithm used, in order to avoid implementations making use of different padding schemes.

The suggested list of padding schemes is to use PKCS-7 padding scheme for symmetric encryption, PKCS-1 and OAEP for assymetric encryption. This should then be reflected in the definitions of the crypto functions in TS 29.198-3.

More work is still needed to specify the format of the input with the IV.

2.3 Security of terminateAccess() function in OSA

2.3.1 no indication of public key/certificate to be used by verifier

No conclusion reached in CN5 on solution to adopt (new parameter or CMS)

2.3.2 no anti-replay protection

No conclusion reached in CN5 on solution to adopt.

2.3.3 no negotiation of signature algorithm

This is related to 2.1.2 above.

2.3.4 specification of signature algorithm

Proposed solution with new algorithms as introduced in contribution seems to be accepted.

2.4 Use of one-way hash function for CHAP in OSA

2.4.1 use of RFC 1994 packet formats

Proposed solution to clarify the use of the PPP packet format and the values of the fields therein seems to be accepted.

2.4.2 weak use of one-way hash function

No final conclusion reached yet due to some confusion around the replacement of MD5 by a MAC/HMAC function and the perception that MAC/HMAC require a symmetric key while MD5 does not (not forgetting that MD5 is here used in a challenge-based response scheme).

3 Conclusion

SA3 should review the attached contributions in light of the above status information. Clear recommendations should be made to CN5 where necessary. It is noted that CN5 is meeting this week in Europe.

3GPP TSG CN5

Apr 8-12, 2002

Sophia Antipolis, FR

Source: Alcatel

Title: Authentication Scheme Negotiation in OSA

Document for: Adoption

Agenda item: T.b.d.

1 Introduction

This contribution discusses the mechanism defined in TS 29.198-3 v4.4.0 to negotiate the authentication scheme used between the client application and the framework/services. A new mechanism is proposed in this contribution to really implement negotiation of authentication mechanisms between the client and the framework/service.

This contribution was originally presented to SA3 (meeting #22) where this was discussed. This contribution is assumed to be in line with the discussions held within SA3.

2 Current mechanism

As per TS 29.198-3, the negotiation of the authentication mechanism is achieved with the initiateAuthenticate() method, which enables the client to indicate which (single) authentication scheme it is willing to use. Currently, two methods have been defined: P_OSA_AUTHENTICATION indicates the use of CHAP (challenge-based authentication with MD5) and P_AUTHENTICATION indicates use of an underlying mechanism (eg CORBA). Other authentication schemes can be defined by service providers and be identified with prefix "SP_".

New authentication schemes under the SP_ prefix are therefore reserved for service providers and would therefore not appear in the standard. Two different service providers may also well assign their own (different) names to the same authentication scheme. This limits the extensibility of the whole mechanism.

In addition, the current mechanism does not enable negotiation of the authentication scheme, since the client indicates a single chosen scheme as a parameter to the initiateAuthentication() method. This limits the scalability of the whole mechanism.

The current specification does not either enable to negotiate the signing algorithm to be used with the terminateAccess() function. A separate contribution discusses this issue further but proposes no solution. We are here proposing a solution in the context of the initial negotiation mechanism.

3 New negotiation mechanism

Several alternative solutions can be designed to solve the above issues. One must be chosen. The attached CR implements alternative 2.

1. The P_OSA_AUTHENTICATION method is extended to apply to any authentication method defined in OSA, not only CHAP_with_MD5. A new method,

selectAuthenticationMethod(), is defined that enables to negotiate which mechanism to use (CHAP_with_MD5, CHAP_with_HMAC_SHA1, digital signature schemes, …). This new method is then used after initiateAuthentication(). With this solution, the selectAuthenticationMethod() function can also be used to negotiate, as a second parameter, the signing algorithm for the terminateAccess().

2. The authType parameter of the initiateAuthentication() method is modified to carry a list of proposed authentication schemes. The return result must then also contain the scheme chosen by the framework. New authentication types are then defined in table TpAuthType to cover other authenticaton schemes such as digital signature-based schemes, use of HMAC with MD5 or SHA1 in CHAP, … With this solution, the signing algorithm for the terminateAccess() function cannot be negotiated except if the authentication scheme negotiated is always a digital signature scheme, which would then also apply to the terminateAccess() function. To be able to negotiate the signing algorithm for terminateAccess() separately, the authType parameter must be made compound to contain two lists of proposals: one for initial authentication and one for the signing algorithm of the terminateAccess() function.

3GPP TSG CN WG5 **Document**

e.g. for 3GPP use the format TP-99xxx
or for SMG, use the format P-99-xxx

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-3** **CR** ⌘ ev **-** ⌘ Current version: **4.4.0** ⌘

*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 ☒

Title:	⌘	Negotiation of Authentication Scheme in OSA
Source:	⌘	Alcatel
Work item code:	⌘	***Date:*** ⌘ 06-04-02
Category:	⌘	**C** ***Release:*** ⌘ Rel-4

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: ⌘ As per TS 29.198-3, the negotiation of the authentication mechanism is achieved with the initiateAuthenticate() method, which enables the client to indicate which (single) authentication scheme it is willing to use. Currently, two methods have been defined: P_OSA_AUTHENTICATION indicates the use of CHAP (challenge-based authentication with MD5) and P_AUTHENTICATION indicates use of an underlying mechanism (eg CORBA). Other authentication schemes can be defined by service providers and be identified with prefix "SP_".

New authentication schemes under the SP_ prefix are therefore reserved for service providers and would therefore not appear in the standard. Two different service providers may also well assign their own (different) names to the same authentication scheme. This limits the extensibility of the whole mechanism.

In addition, the current mechanism does not enable negotiation of the authentication scheme, since the client indicates a single chosen scheme as a parameter to the initiateAuthentication() method. This limits the scalability of the whole mechanism.

The current specification does not either enable to negotiate the signing algorithm to be used with the terminateAccess() function.

Summary of change: ⌘ The authType parameter of the initiateAuthentication() method is modified to carry a list of proposed authentication schemes. The return result must then also contain the scheme chosen by the framework. New authentication types are then defined in table TpAuthType to cover other authenticaton schemes such as digital signature-based schemes, use of HMAC with MD5 or SHA1 in CHAP, … With this solution, the signing algorithm for the terminateAccess() function cannot be negotiated except if the authentication scheme negotiated is always a digital signature scheme, which would then also apply to the terminateAccess() function. To be able to do so, the authType parameter is made compound to contain two lists of proposals: one for initial authentication and one for the signing algorithm of the terminateAccess() function.

Consequences if not approved:	⌘	Restrictions on possibilities to extend OSA with new standard authentication schemes and no negotiation of signature function in terminateAccess().

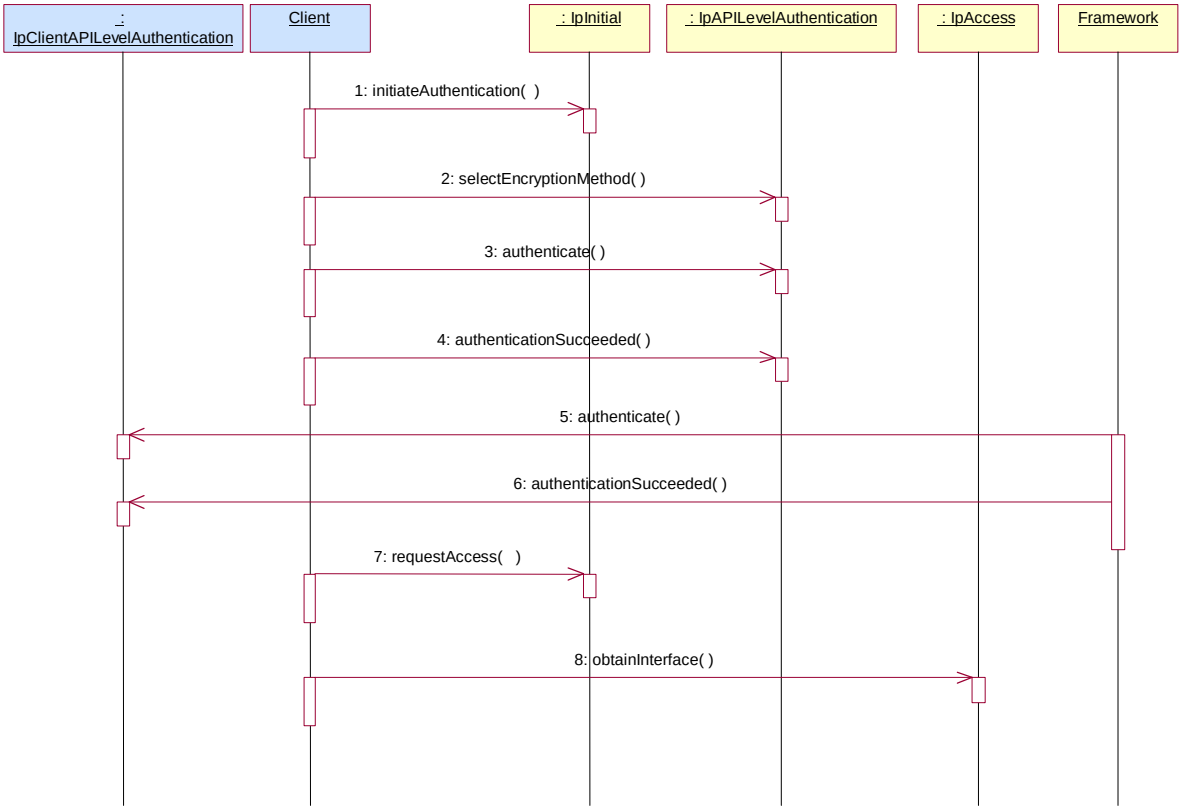
Clauses affected:	⌘			
Other specs affected:	⌘	☐ Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	
Other comments:	⌘			

4.1.1.1 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.



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.

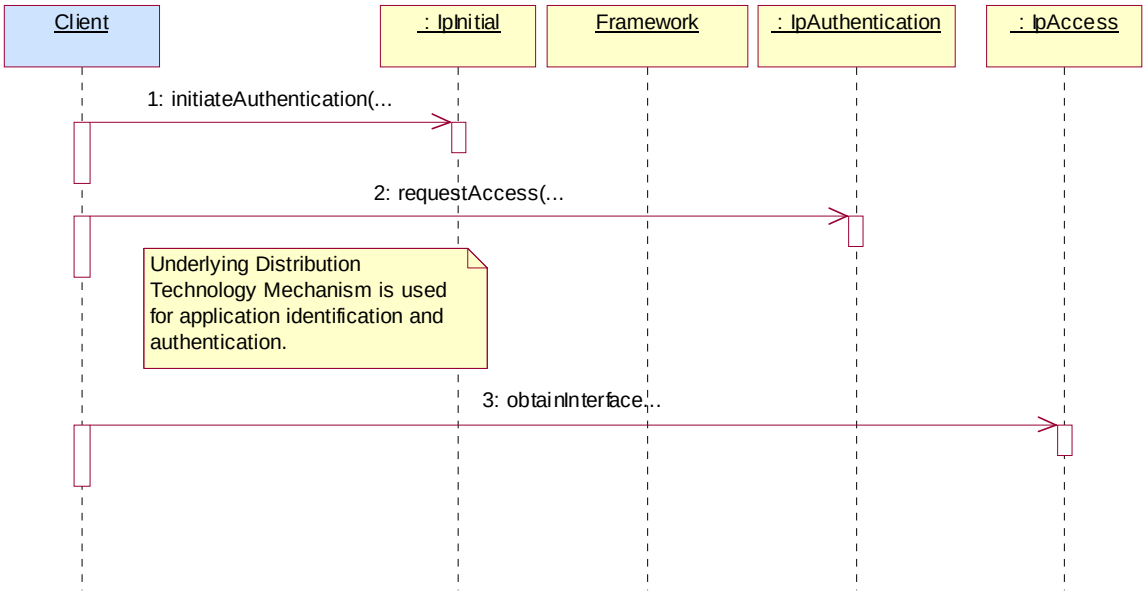
7: Request Access

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 invokes obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

4.1.1.2 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 can now invoke obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

4.1.1.3 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.

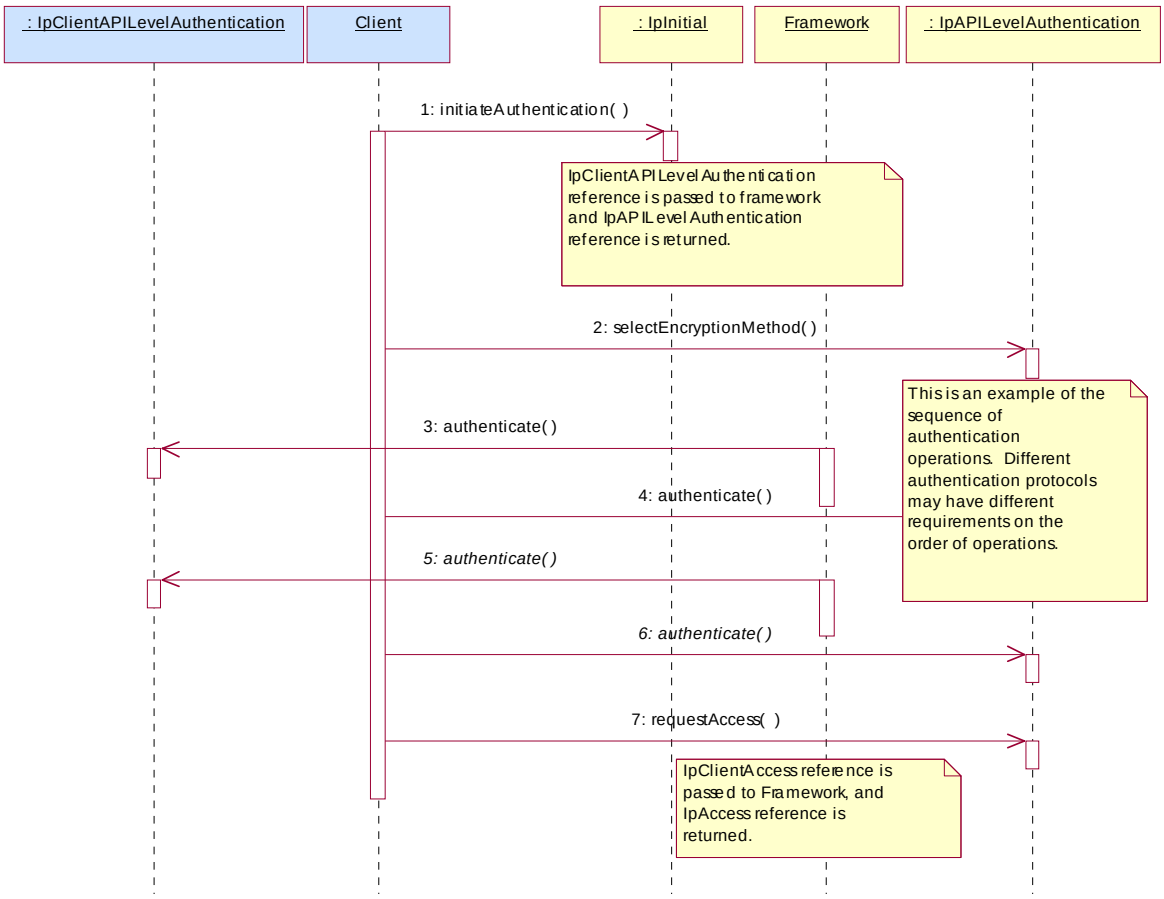
The client must authenticate with the Framework before it is able to use any of the other interfaces supported by the Framework. Invocations on other interfaces will fail until authentication has been successfully completed.

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 according to RFC 1994 CHAP specification. 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.



4.2 Class Diagrams

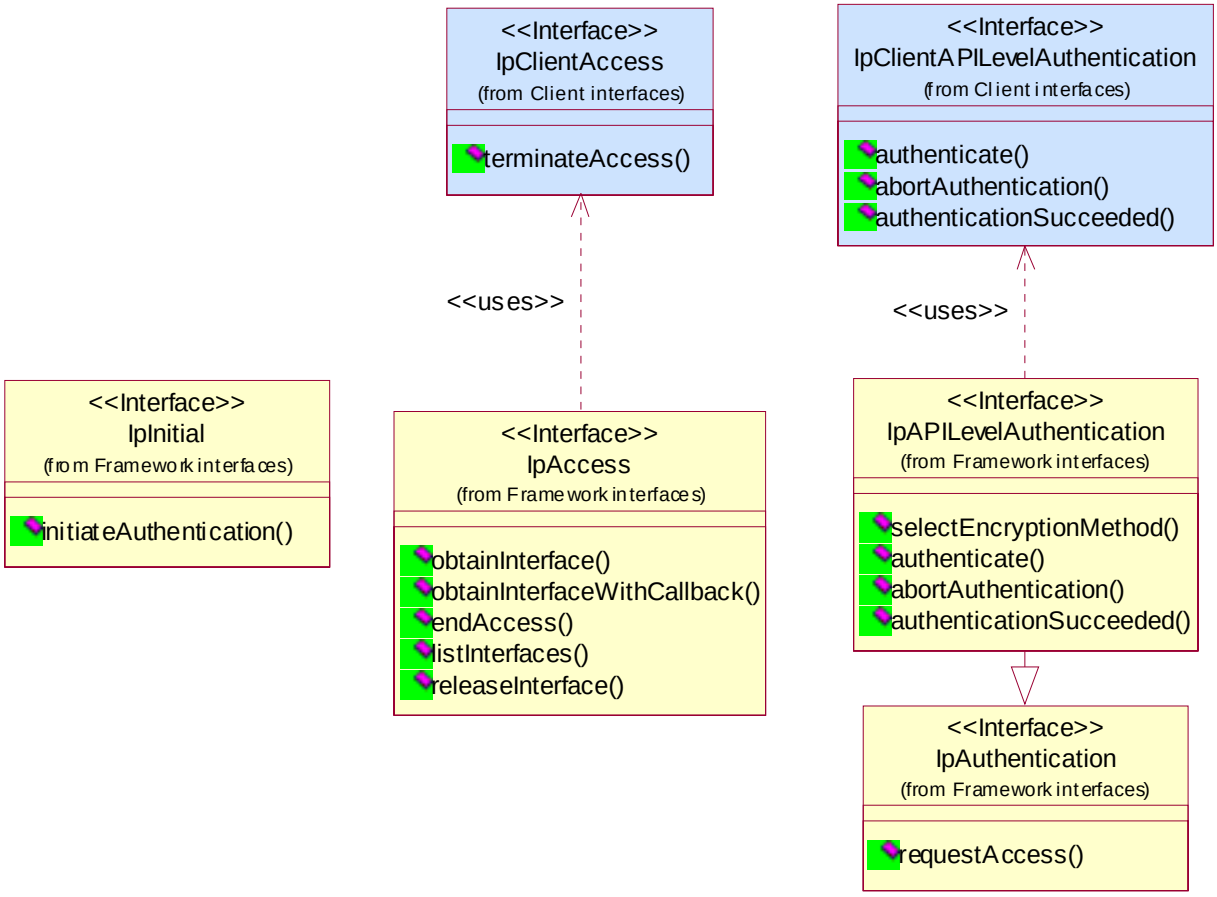


Figure: Trust and Security Management Package Overview

~~4.1.1.1~~4.2.1.1 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
initiateAuthentication (clientDomain : in TpAuthDomain, authType : in TpAuthTypeList) : TpAuthDomainAndAuthType

Method
`initiateAuthentication()`

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 existing registered service (i.e. TpServiceID) 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`<u>**`List`**</u>

This identifies the type<u>s</u> of authentication mechanism<u>s</u> ~~requested~~ <u>supported</u> by the client. It provides operators and clients with the opportunity to <u>negotiate which authentication method and also to</u> 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`<u>**`AndAuthType`**</u>

```
structure TpAuthDomainAndAuthType {
    fwDomain:   TpAuthDomain;
    authType: TpAuthType;
}
```

Raises

TpCommonExceptions, P_INVALID_DOMAIN_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_AUTH_TYPE

~~4.1.2~~4.2.2 Trust and Security Management State Transition Diagrams

~~4.1.1.1~~4.2.2.1 State Transition Diagrams for IpInitial

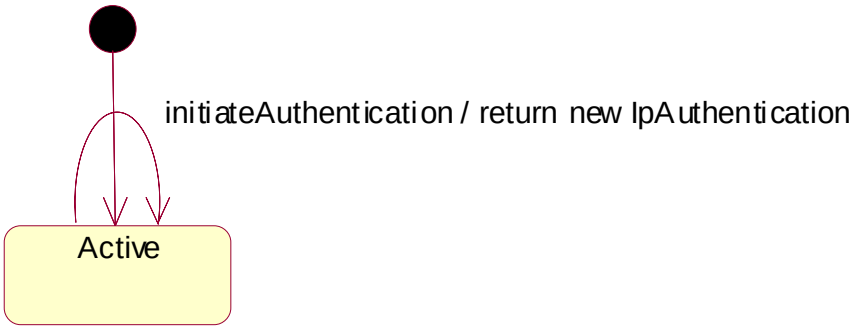


Figure : State Transition Diagram for IpInitial

~~4.1.1.1.1~~4.2.2.1.1 Active State

~~4.1.1.2~~4.2.2.2 State Transition Diagrams for IpAPILevelAuthentication

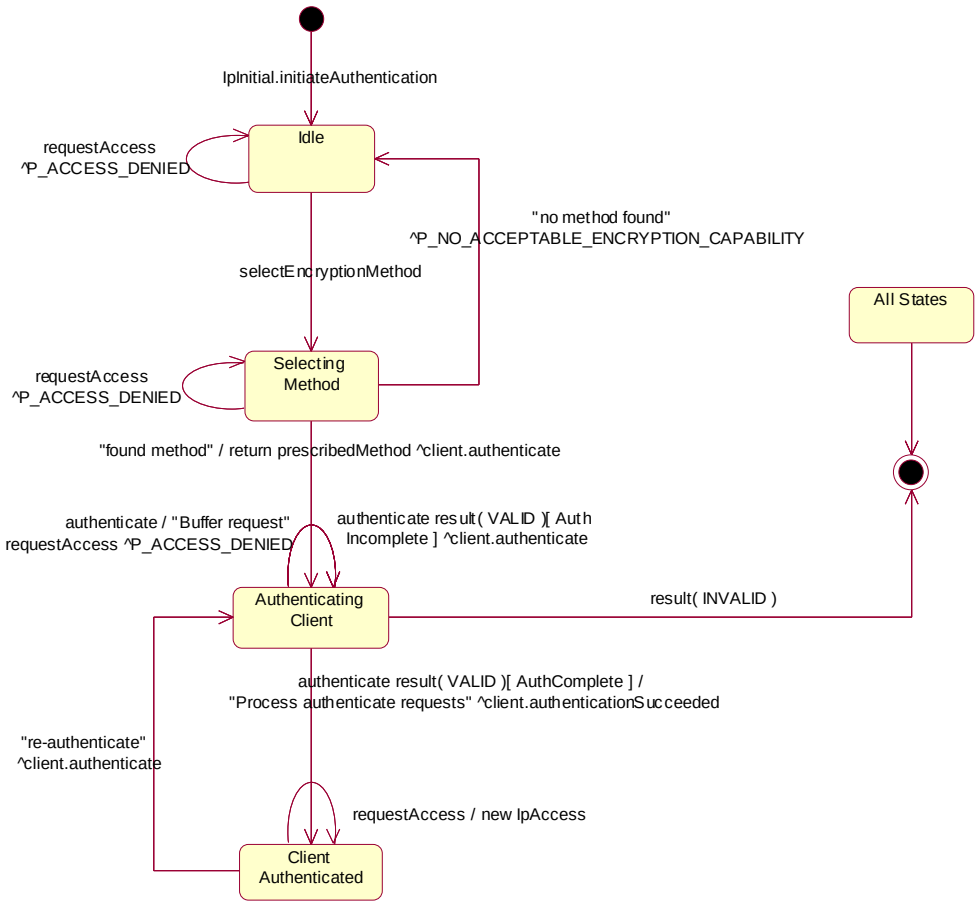


Figure : State Transition Diagram for IpAPILevelAuthentication

~~4.1.1.1.1~~4.2.2.2.1 Idle State

When the client has invoked the IpInitial initiateAuthentication method, an object implementing the IpAPILevelAuthentication interface is created. The client now has to provide its encryption capabilities by invoking selectEncryptionMethod.

~~4.1.1.1.2~~4.2.2.2.2 Selecting Method State

In this state the Framework selects the preferred encryption mechanism within the capability of the client. It is a policy of the framework (perhaps agreed off-line with the enterprise operator) whether the client has to be authenticated or not. In case no mechanism can be found the P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY exception is thrown and the Authentication object moves back to the IDLE state The client can now revisit its list of supported capabilities to identify whether it is complete. If it has no more encryption capabilities to use, then it must invoke abortAuthentication.

~~4.1.1.1.3~~4.2.2.2.3 Authenticating Client State

When entering this state, the Framework requests the client to authenticate itself by invoking the Authenticate method on the client. In case the client requests the Framework to authenticate itself by invoking Authenticate on the IpAPILevelAuthentication interface, the Framework will either buffer the requests and respond when the client has been authenticated, or respond immediately, depending on policy. When the Framework has processed the response from the Authenticate request on the client, the response is analysed. If the response is valid but the authentication process is not yet complete, then another Authenticate request is sent to the client. If the response is valid and the authentication process has been completed, then a transition to the state ClientAuthenticated is made, the client is informed of its success by invoking authenticationSucceeded, then the framework begins to process any buffered authenticate requests. In case the response is not valid, the Authentication object is destroyed. This implies that the

client has to re-initiate the authentication by calling once more the initiateAuthentication method on the IpInitial interface.

~~4.1.1.1.4~~4.2.2.2.4 Client Authenticated State

In this state the client is considered authenticated and is now allowed to request access to the IpAccess interface. In case the client requests the Framework to authenticate itself by invoking Authenticate on the IpAPILevelAuthentication interface, the Framework provides the correct response to the challenge. If the framework decides to re-authenticate the client, then the authenticate request is sent to the client and a transition back to the AuthenticatingClient state occurs.

4.3 Class Diagrams

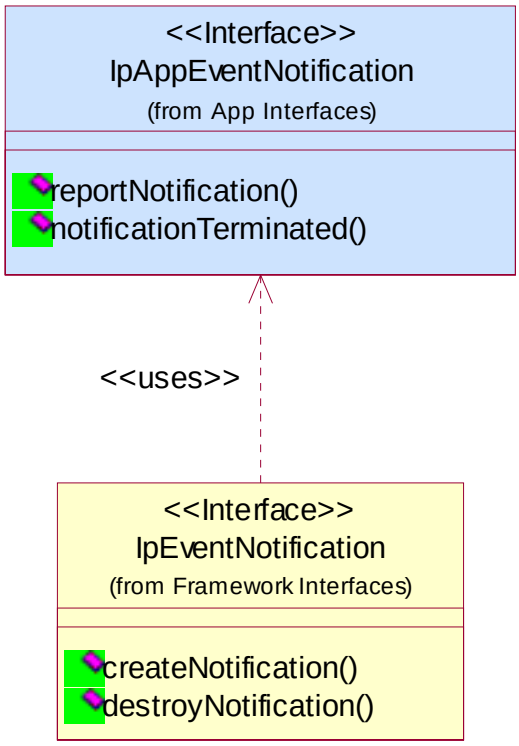


Figure: Event Notification Class Diagram

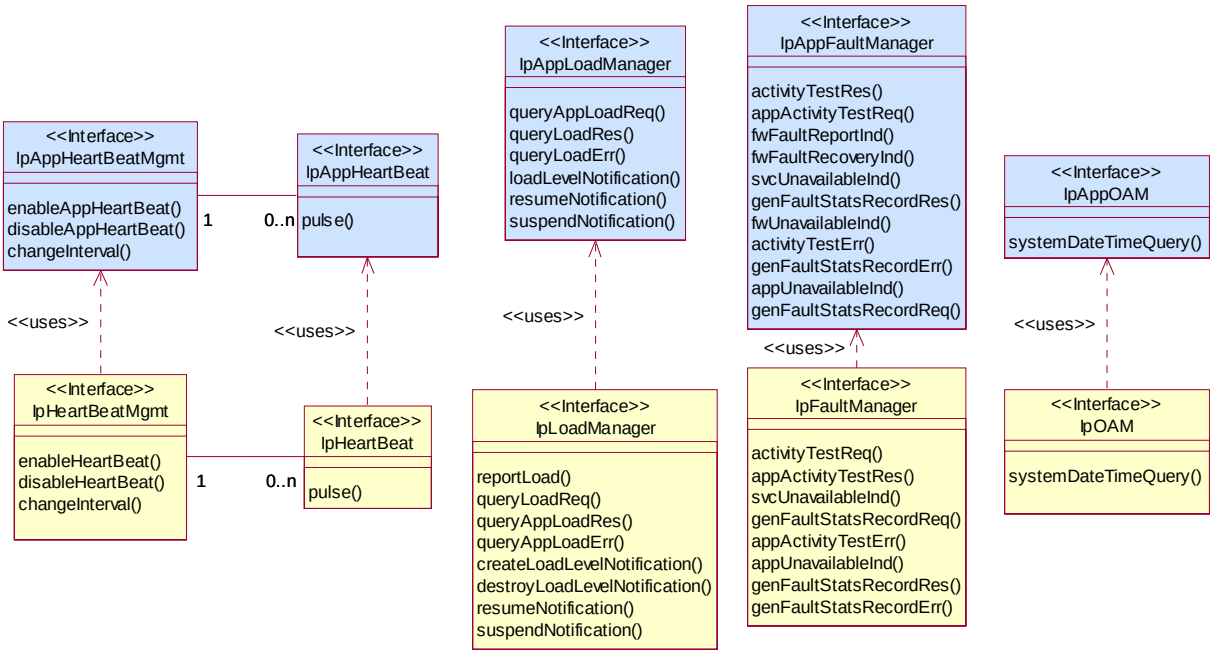


Figure: Integrity Management Package Overview

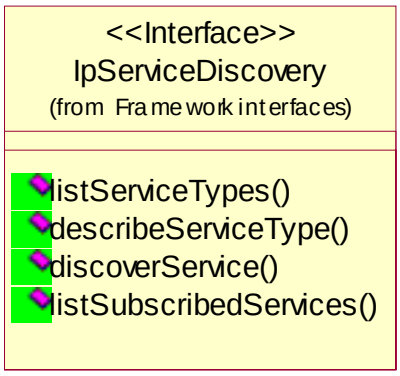


Figure: Service Discovery Package Overview

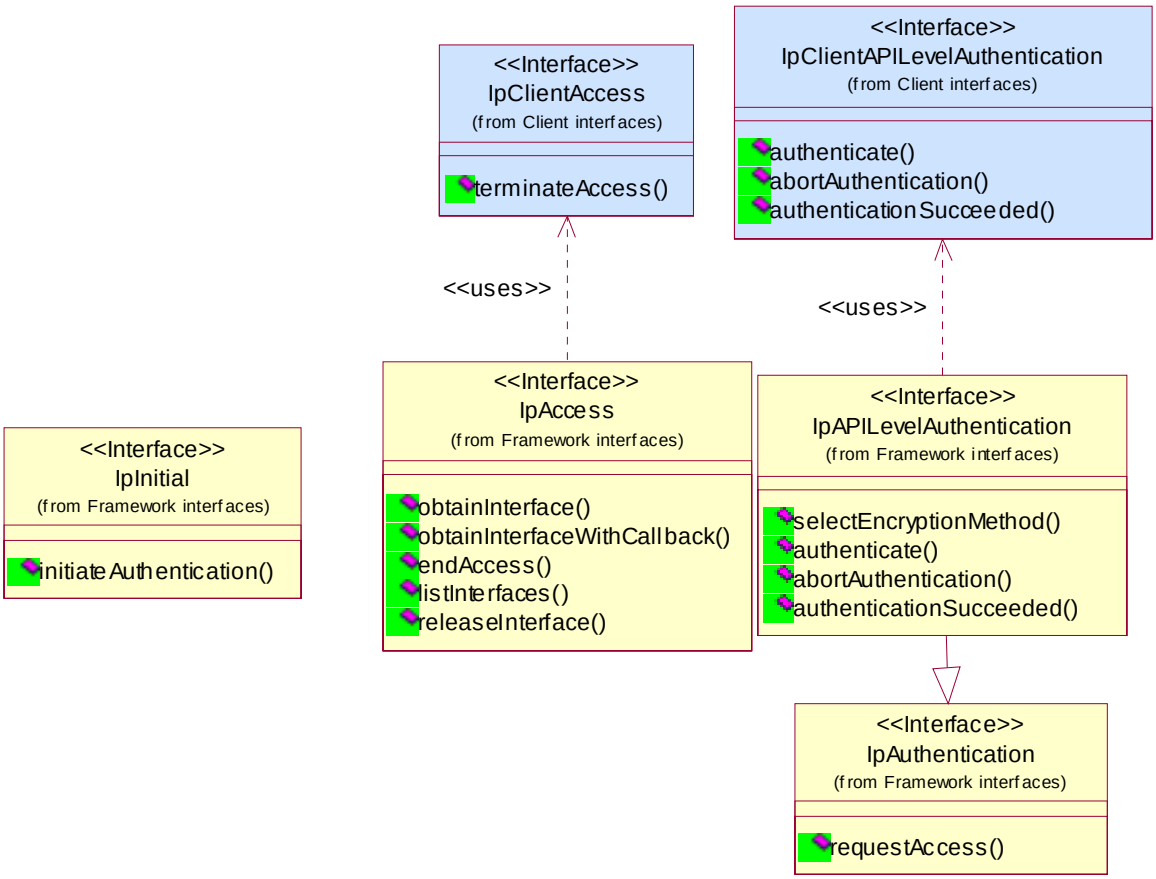


Figure: Trust and Security Management Package Overview

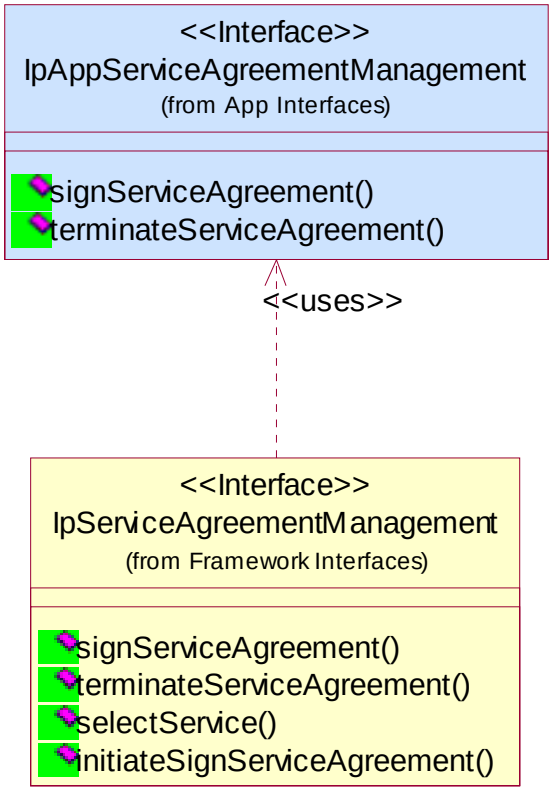


Figure: Service Agreement Management Package Overview

4.4 Class Diagrams

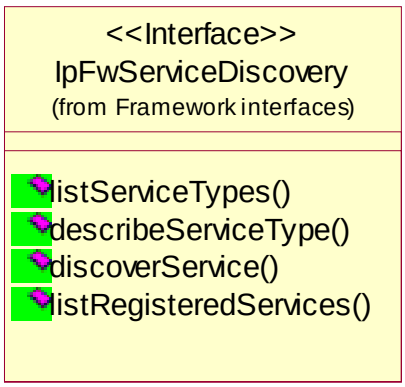


Figure: Service Discovery Package Overview

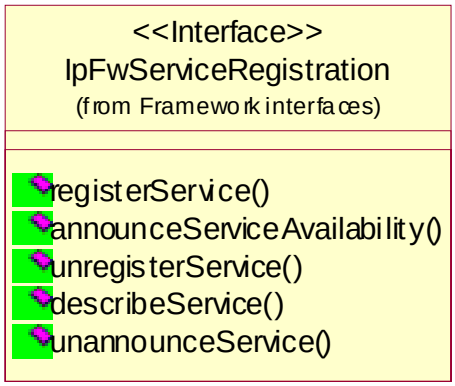


Figure: Service Registration Package Overview

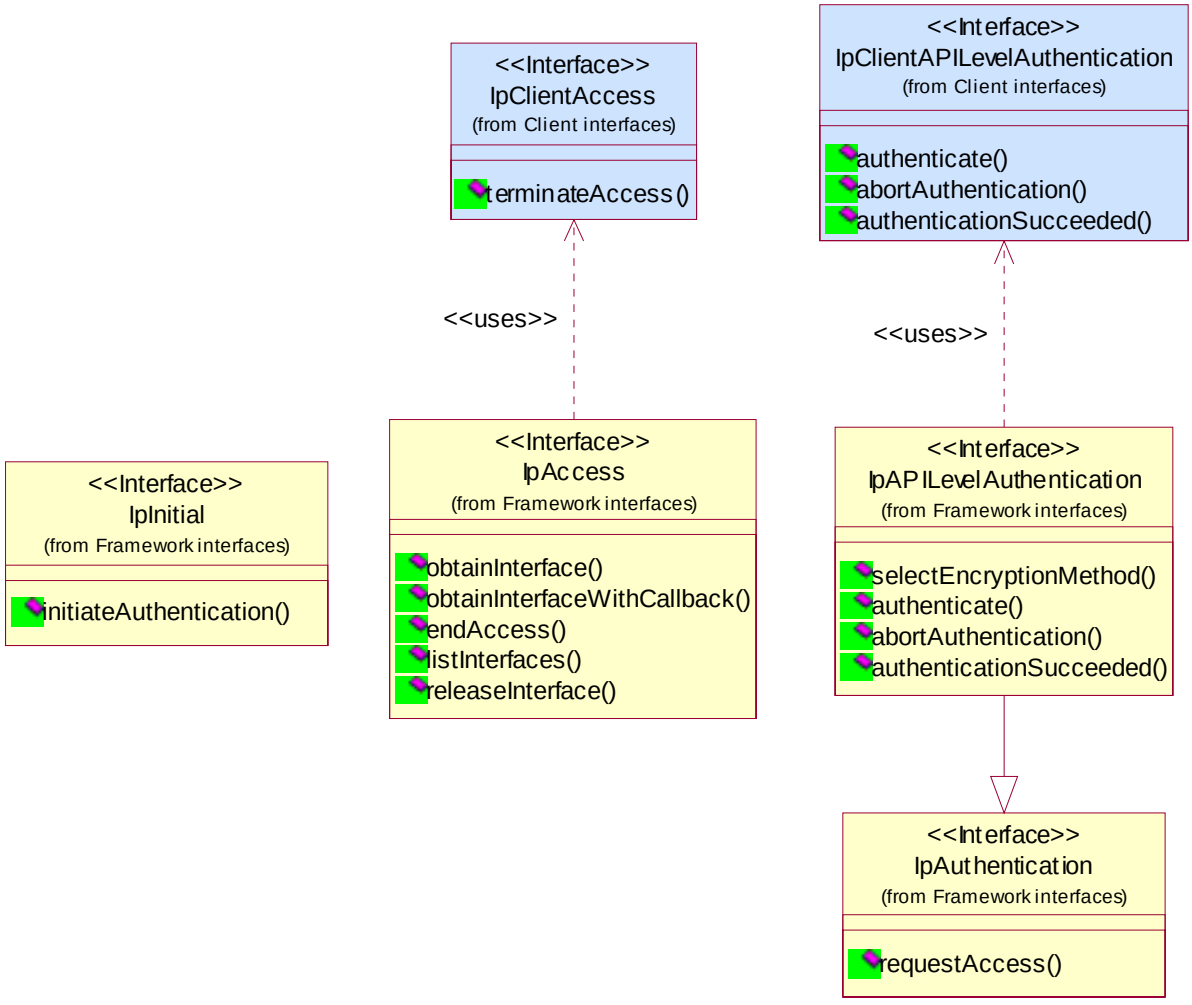


Figure: Trust and Security Management Package Overview

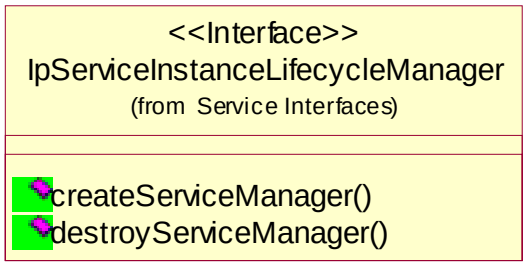


Figure: Service Instance Lifecycle Manager Package Overview

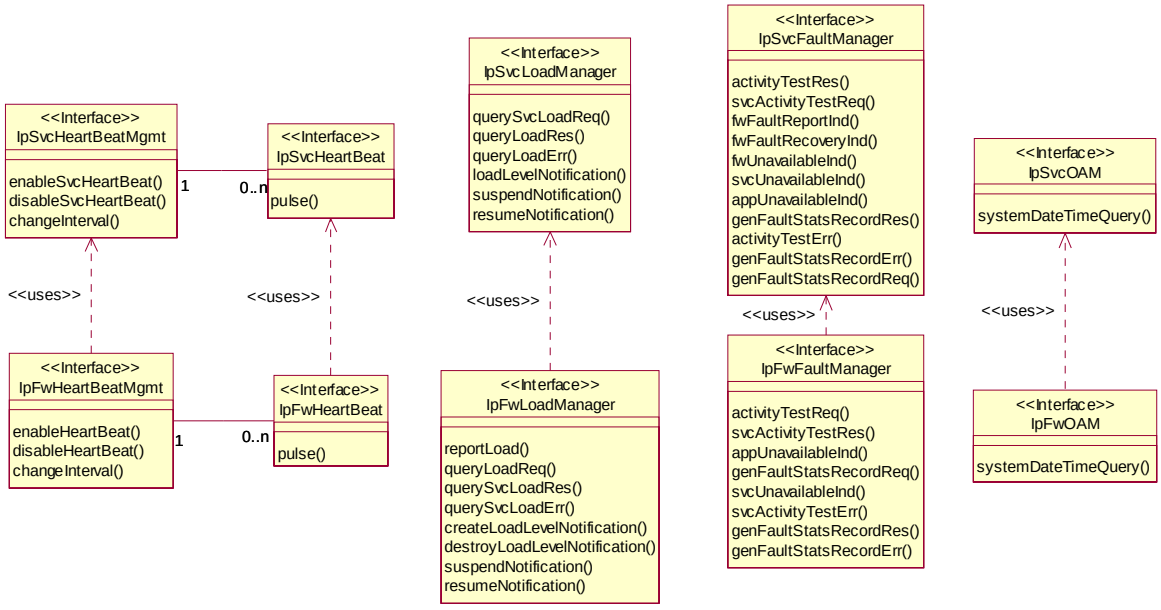


Figure: Integrity Management Package Overview

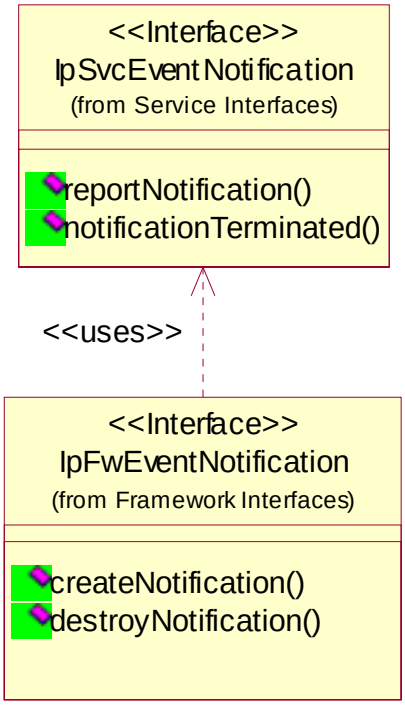


Figure: Event Notification Package Overview

~~4.1.1~~4.4.1 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. Authentication is based on RFC 1994 CHAP mechanism using MD5 hashing algorithm
P_AUTHENTICATION	Authenticate using the implementation specific authentication mechanism, e.g. CORBA Security.
P_OSA_HMAC_SHA1	Authenticate using the OSA API Level Authentication Interfaces: IpAPILevelAuthentication and IpClientAPILevelAuthentication. Authentication is based on the use of HMAC-SHA1 hashing algorithm to generate a response based on a shared secret and a challenge received via authenticate() method.

3GPP TSG CN WG5– CN5

Sophia Antipolis, FR

Source: Alcatel

Title: Encryption of challenge in CHAP-based OSA authentication

Document for: Adoption

Agenda item: T.b.d.

1 Introduction

This contribution discusses two issues related to a specific functionality in TS 29.198-3 v4.2.0 which makes the challenge used for CHAP-based authentication to be encrypted when passed from the verifier to the claimant.

This is based on a contribution originally discussed at the last SA3 meeting and is expected to reflect these discussions.

2 Issue

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.

The overall authentication phase works as follows:

- the client first uses the initiateAuthenticate() method to set the P_OSA_AUTHENTICATION scheme (ie CHAP).
- with the selectEncryption() method, the client application and the framework agree on a symmetric encryption function to be used to encrypt the challenge sent from the verifier to the claimant.
- the framework can then use the authenticate() method to pass an encrypted challenge string to the client, using the encryption algorithm (DES, triple DES) negotiated in the previous step. Encryption of the challenge string is done thanks to a secret key which must a priori be shared between the client and the framework (out of scope). The client must then decrypt the received encrypted challenge and generate a response based on the decrypted challenge and a secret shared with the framework. The client can authenticate the framework using the exactly same mechanism in the other direction.

We hereby discuss two issues related to the above procedure.

2.1 Issue 1: the need for encrypting the challenge

A fundamental question is whether there is any real security gain in encrypting the challenge string itself. This indeed requires extra management (shared secret key for encryption/decryption between the client and the framework) and processing, while no identified security weakness is solved by this extra encryption process.

We believe that there is no need to have this challenge encryption phase, which should be removed from the authentication procedure[1].

2.2 Issue 2: no formatting defined for challenge encryption.

In the case we still consider the challenge encryption procedure itself, we note that the specification lacks details which make it unimplementable as is.

Symmetric encryption mechanisms such as DES, 3DES, … to be used for challenge encryption require the use of an Initialisation Vector (IV) as input into the encryption/decryption phases. This IV must be passed from the encryptor to the decryptor (or at least known by the decryptor).

The length of the challenge string is not necessarily a multiple of the encryption algorithm block length (eg 8 bytes for DES or 3DES). When it is not the case, padding bytes must be appended to the input (ie challenge string) of the encryption algorithm. After decryption, it is obviously necessary to be able to isolate those padding bytes so as not to use them as part of the challenge string. To avoid potential attacks, it is also important to provide the length of the challenge string within the encrypted data.

The description of the authenticate() method does not cover those aspects.

3 Solution

As presented above, it is suggested to suppress the requirement for encryption of the challenge in the authentication phase. The accompanying proposed CR implements the required modifications to TS 29.198-3 v4.4.0 by removing the selectEncryptionMethod() from the specification, since its sole purpose is to negotiate the encryption mechanism used in challenge-based authentication.

Use of public-key based authentication schemes should be specified separately and not on top of CHAP itself.

---

[1] It is noted that this view was shared by SA3 delegates during the joint SA3-CN5 meeting held in Bristol on Feb 25th.

3GPP TSG CN WG5 **Document**

e.g. for 3GPP use the format TP-99xxx
or for SMG, use the format P-99-xxx

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-3** **CR** ⌘ ev **-** ⌘ Current version: **4.4.0** ⌘

*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 ☒

Title:	⌘	Encryption of challenge in CHAP-based OSA authentication
Source:	⌘	Alcatel
Work item code:	⌘	***Date:*** ⌘ 06-04-02
Category:	⌘	**F** ***Release:*** ⌘ Rel-4

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.

The overall authentication phase works as follows:

the client first uses the initiateAuthenticate() method to set the P_OSA_AUTHENTICATION scheme (ie CHAP).

with the selectEncryption() method, the client application and the framework agree on a symmetric encryption function to be used to encrypt the challenge sent from the verifier to the claimant.

the framework can then use the authenticate() method to pass an encrypted challenge string to the client, using the encryption algorithm (DES, triple DES) negotiated in the previous step. Encryption of the challenge string is done thanks to a secret key which must a priori be shared between the client and the framework (out of scope). The client must then decrypt the received encrypted challenge and generate a response based on the decrypted challenge and a secret shared with the framework. The client can authenticate the framework using the exactly same mechanism in the other direction.

A fundamental question is whether there is any real security gain in encrypting the challenge string itself. This indeed requires extra management (shared secret key for encryption/decryption between the client and the framework) and processing, while no identified security weakness is solved by this extra encryption process. There is no need to have this challenge encryption phase, which should be removed from the authentication procedure.

Summary of change: ⌘ It is suggested to suppress the requirement for encryption of the challenge in the authentication phase. This CR implements the required modifications to TS 29.198-3 v4.4.0 by removing the selectEncryptionMethod() from the specification, since its sole

		purpose is to negotiate the encryption mechanism. The TpEncryption tables are also removed. The fact that public key-based authentication mechanisms could be used is clarified in the authenticate) function.
Consequences if not approved:	⌘	Unnecessary use of an encryption mechanism complexifies the system without any gain. Also, lack of details of encryption procedure will lead to interoperability issues.

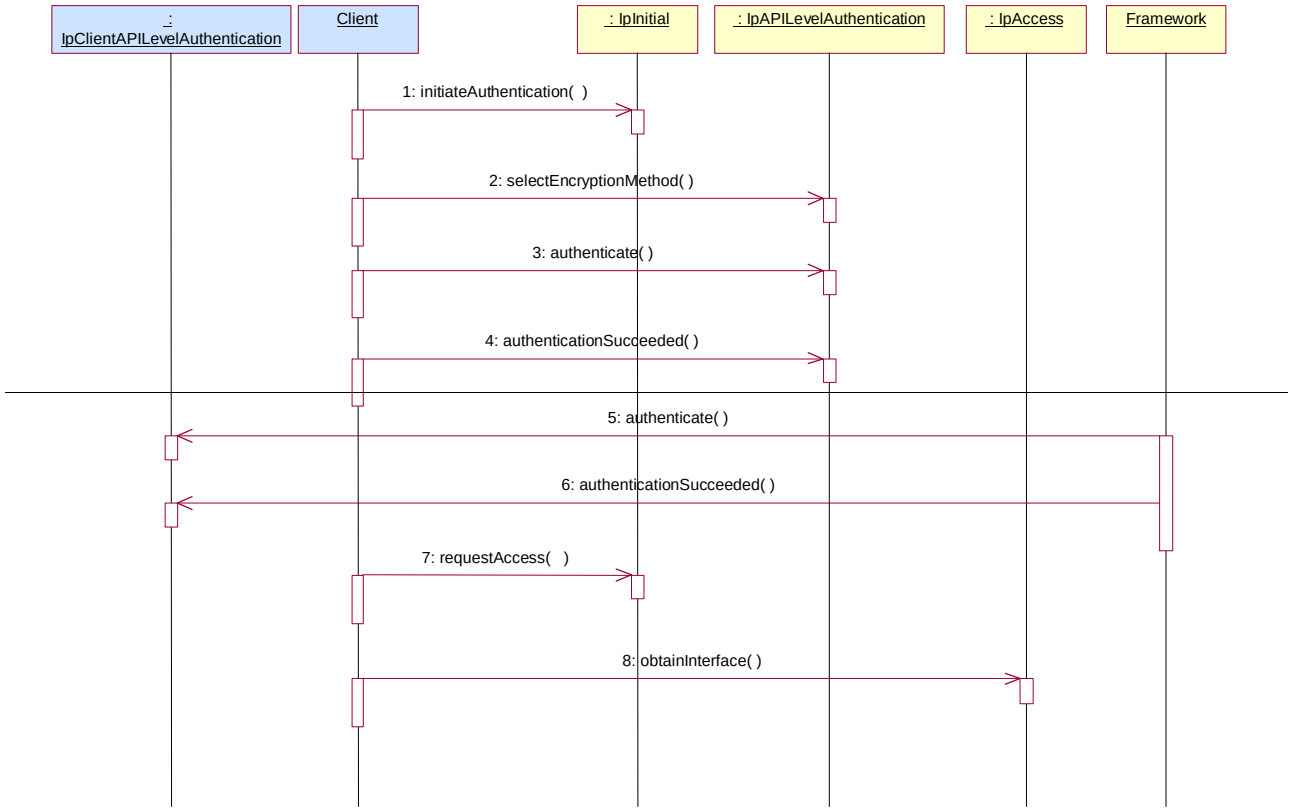
Clauses affected:	⌘	4.1.1.1, 4.1.1.2, 4.1.1.3, 4.2, 6.3.1.1, 6.3.1.5, 10.3.3, 10.3.4		
Other specs affected:	⌘	☐ Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	
Other comments:	⌘			

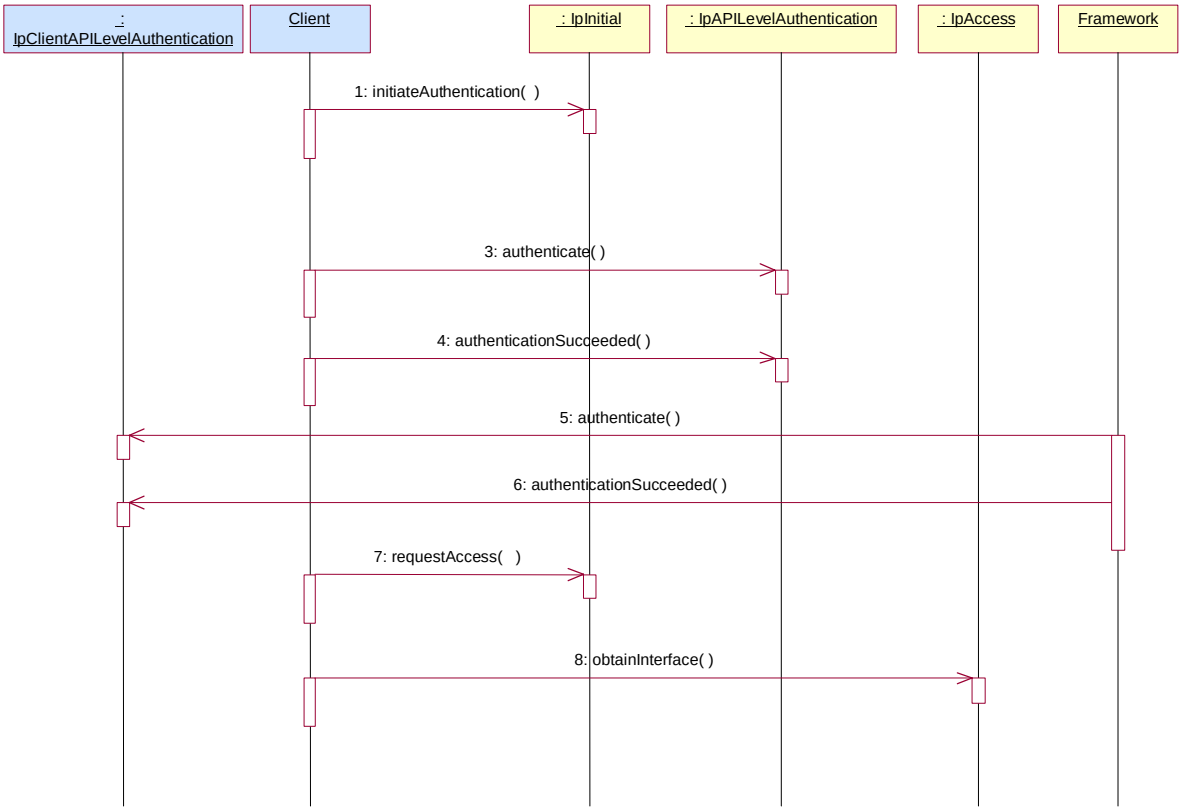
4.1.1.1 Initial Access

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.





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.~~

2~~3~~: Authenticate

3~~4~~: The client provides an indication if authentication succeeded.

4~~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.

5~~6~~: The Framework provides an indication if authentication succeeded.

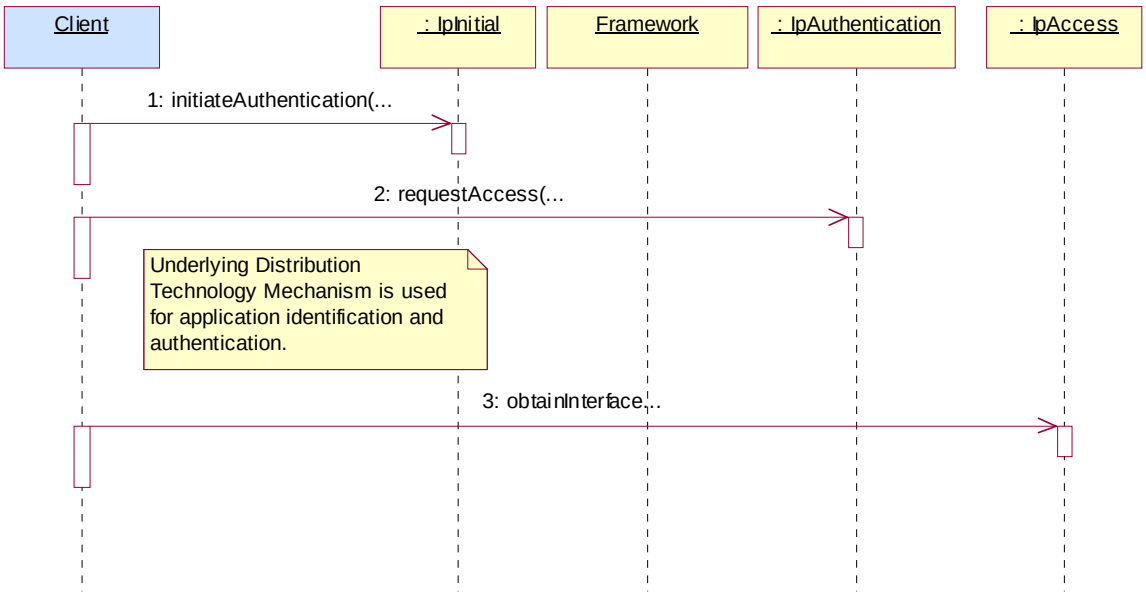
6~~7~~: Request Access

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.

7~~8~~: The client invokes obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

4.1.1.2 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 can now invoke obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

4.1.1.3 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, challenge-based methods, …) 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.~~

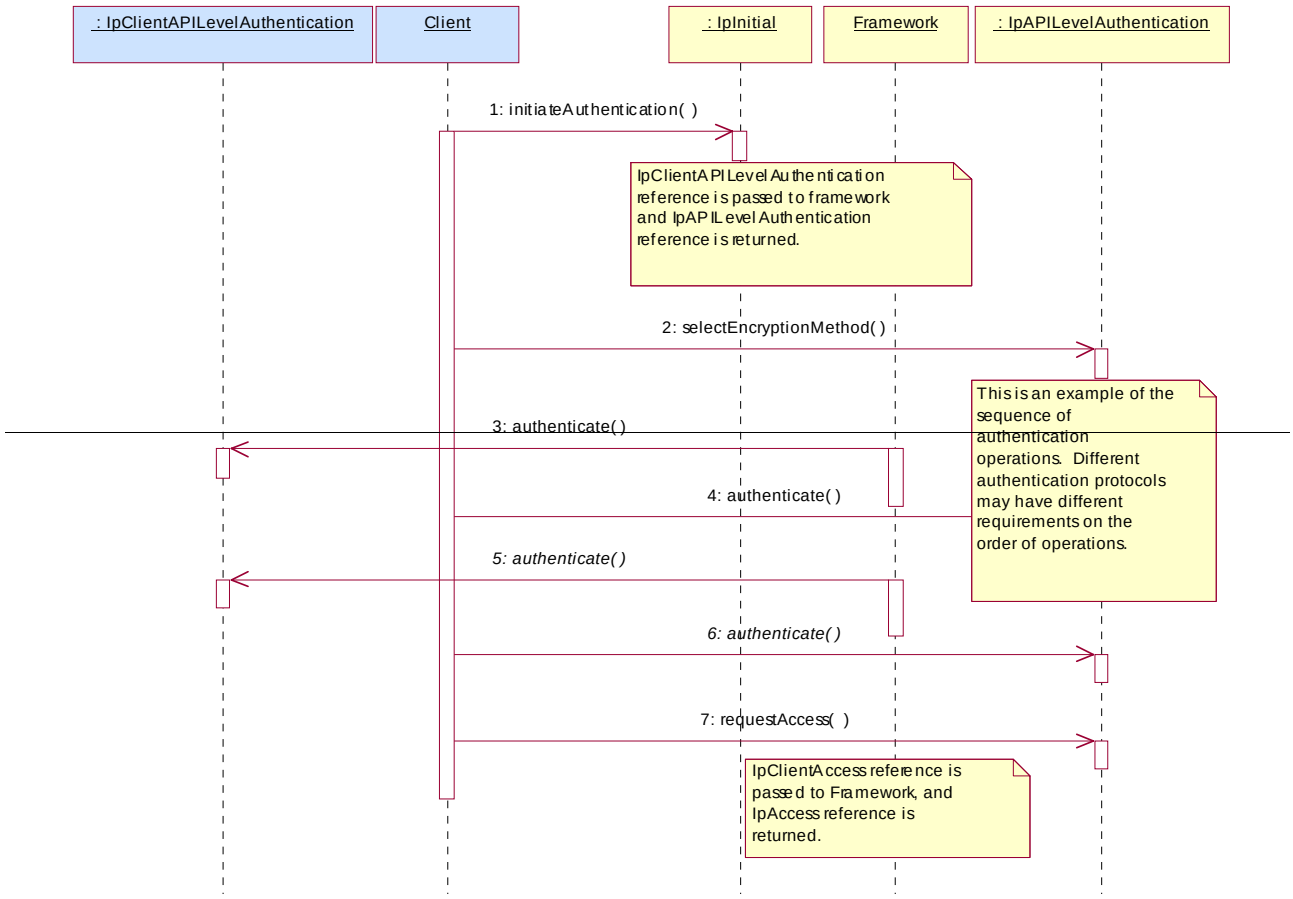
The client must authenticate with the Framework before it is able to use any of the other interfaces supported by the Framework. Invocations on other interfaces will fail until authentication has been successfully completed.

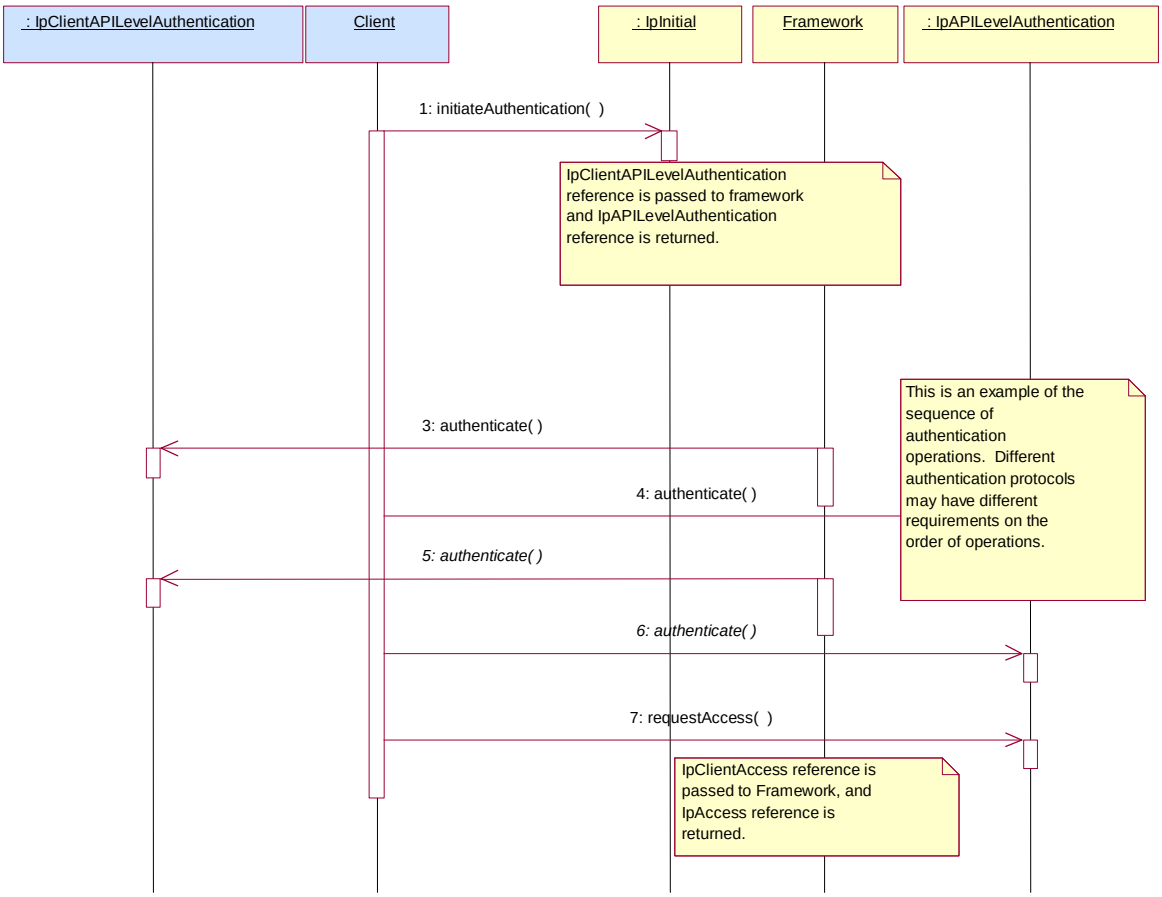
1) The client calls initiateAuthentication on the OSA Framework Initial interface. This allows the client to specify the type of authentication process. This authentication process may be specific to the provider, or the implementation technology used. The initiateAuthentication method can be used to 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.~~

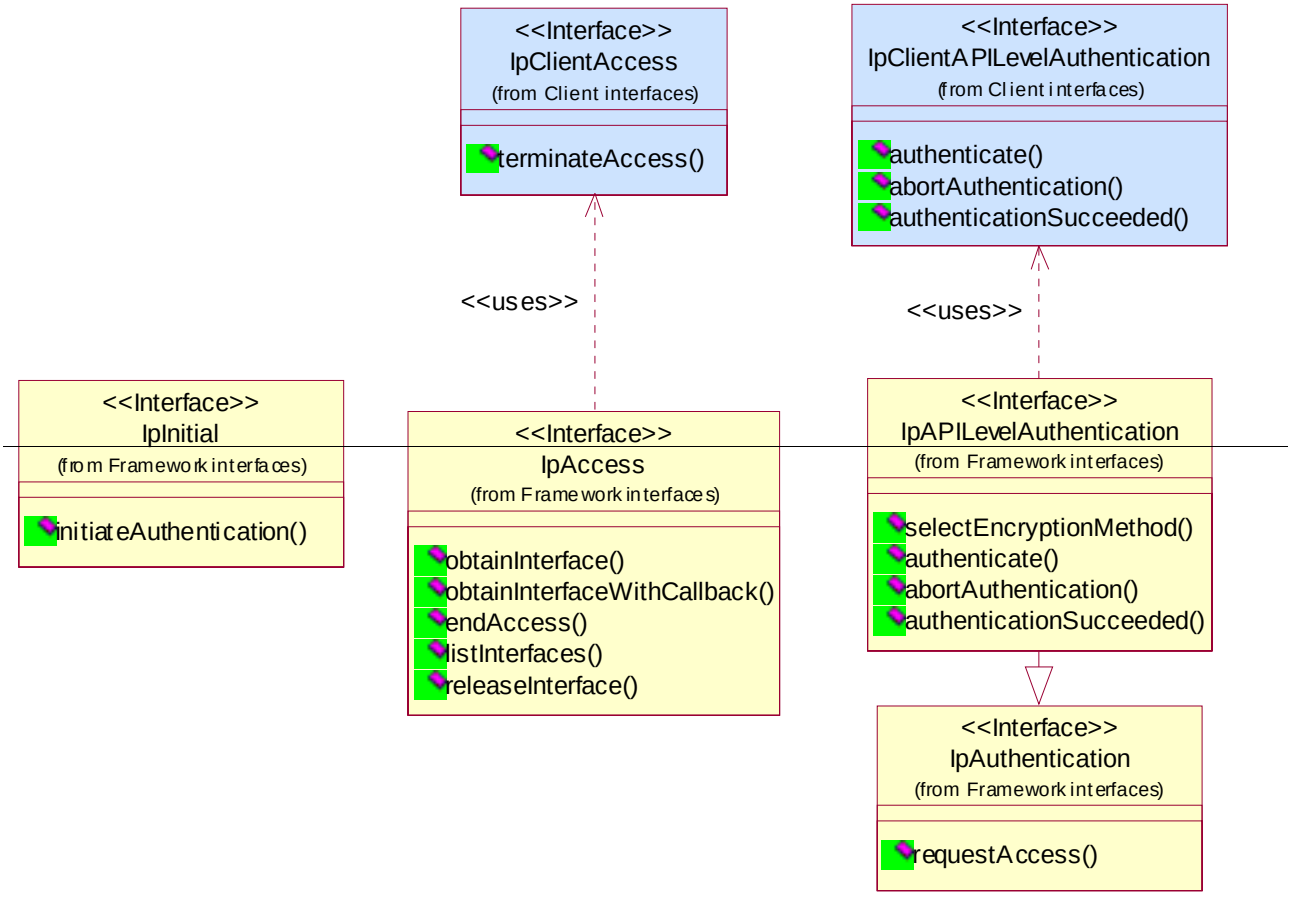
2~~3~~) The application and Framework interact to authenticate each other. For an authentication method of P_OSA_AUTHENTICATION, this procedure consists of a number of challenge/ response exchanges. This authentication protocol is performed using the authenticate method on the API Level Authentication 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( )
This is an example of the sequence of authentication operations. Different authentication protocols may have different requirements on the order of operations.
3: authenticate( )
4: authenticate( )
5: authenticate( )
6: authenticate( )
7: requestAccess( )
IpClientAccess reference is passed to Framework, and IpAccess reference is returned.



4.2 Class Diagrams


<<Interface>>
IpClientAccess
(from Client interfaces)
terminateAccess()
<<Interface>>
IpClientAPILevelAuthentication
(from Client interfaces)
authenticate()
abortAuthentication()
authenticationSucceeded()
<<uses>>
<<uses>>
<<Interface>>
IpInitial
(from Framework interfaces)
initiateAuthentication()
<<Interface>>
IpAccess
(from Framework interfaces)
obtainInterface()
obtainInterfaceWithCallback()
endAccess()
listInterfaces()
releaseInterface()
<<Interface>>
IpAPILevelAuthentication
(from Framework interfaces)
selectEncryptionMethod()
authenticate()
abortAuthentication()
authenticationSucceeded()
<<Interface>>
IpAuthentication
(from Framework interfaces)
requestAccess()

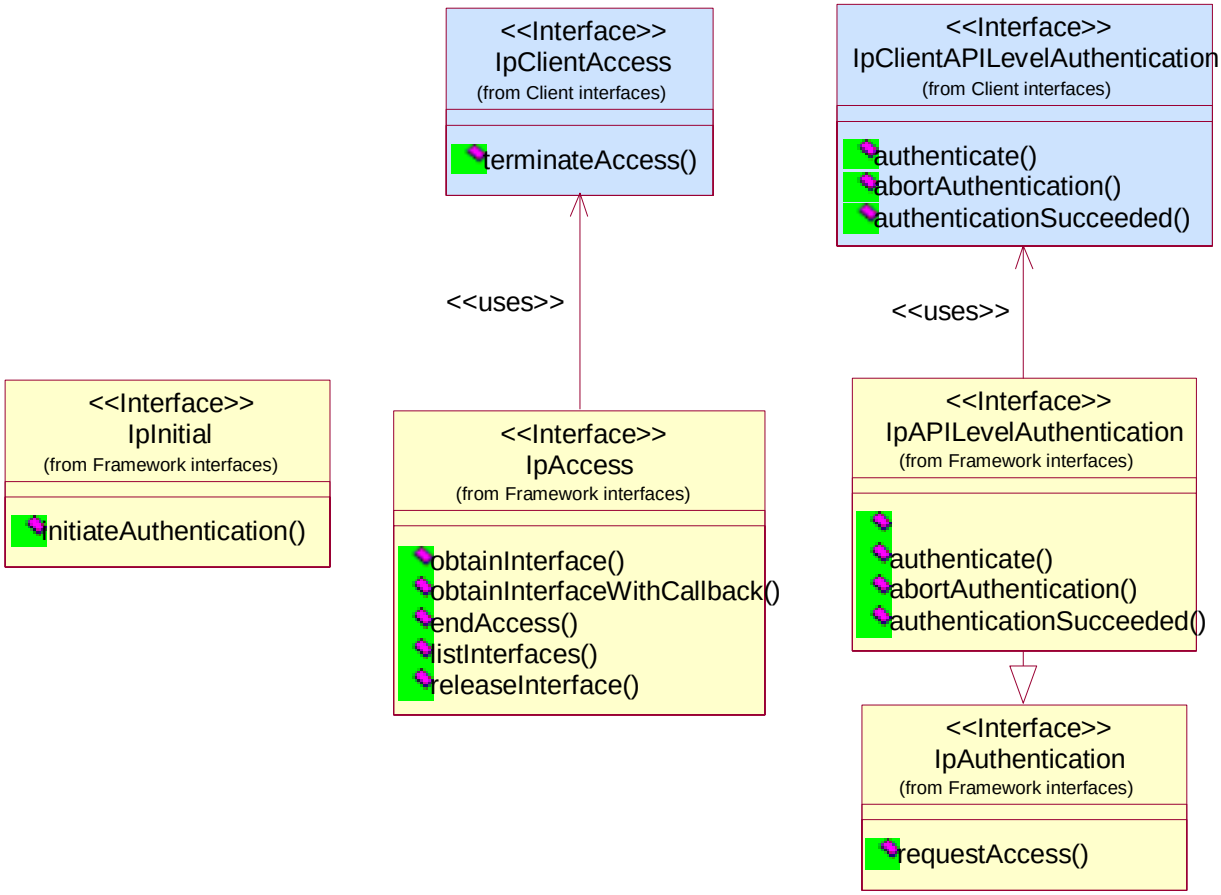
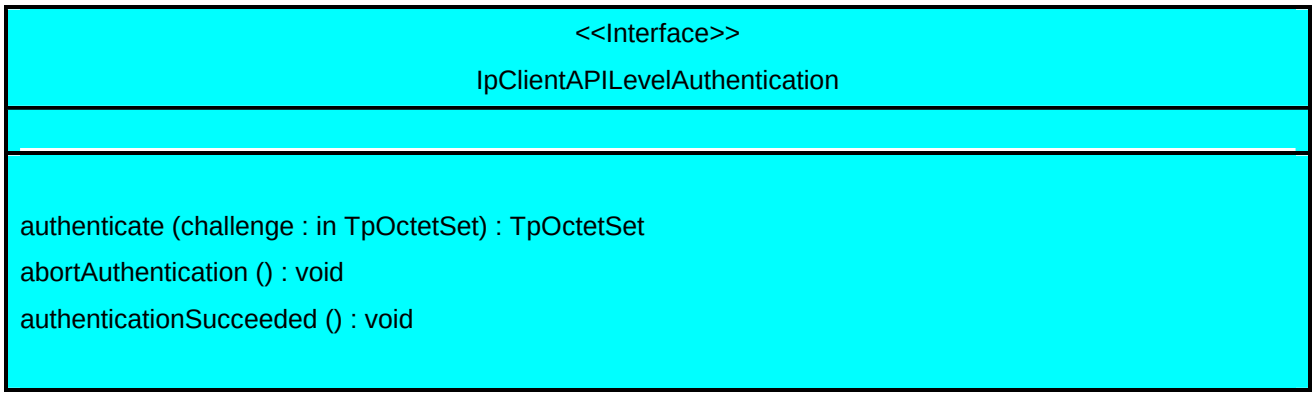


Figure: Trust and Security Management Package Overview

6.3.1.1 Interface Class IpClientAPILevelAuthentication

Inherits from: IpInterface.



Method
authenticate()

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. When a public key-based authentication scheme is used, 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 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. If the authentication method in use is CHAP-based, t~~T~~he 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.

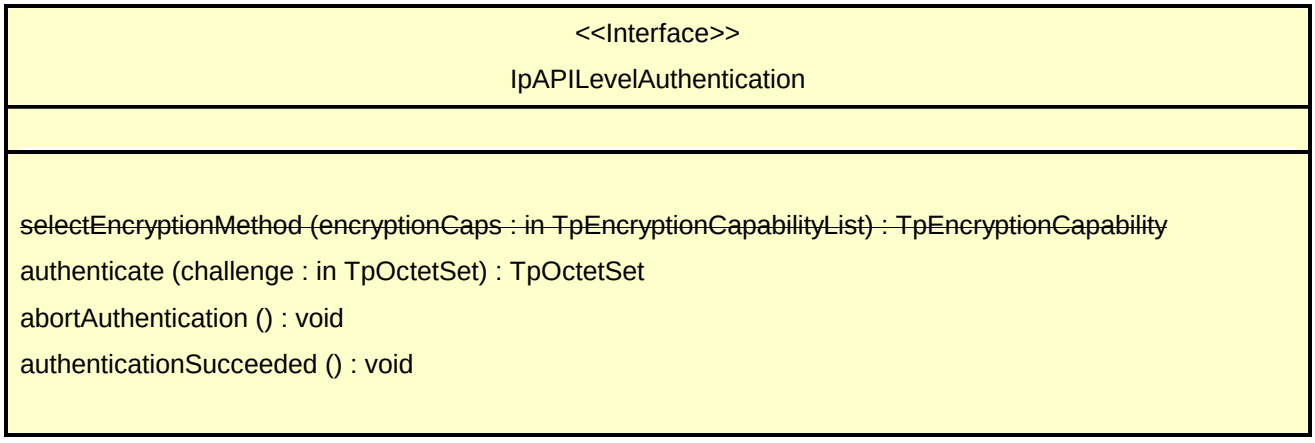
Parameters

No Parameters were identified for this method

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.



Method

selectEncryptionMethod()

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 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. When a public key-based authentication scheme is used, tThe 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. If the authentication method in use is CHAP-based, t~~T~~he 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

10.3.3TpEncryptionCapability

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 1024-bit keys.

10.3.3TpEncryptionCapabilityList

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

3GPP TSG CN WG5

April 8-12, 2002

Sophia Antipolis, FR

Source: Alcatel

Title: Security of terminateAccess() function in OSA

Document for: Adoption

Agenda item: T.b.d.

1 Introduction

This contribution identifies various issues in TS 29.198-3 v4.4.0 with regards to the security mechanism used to protect the terminateAccess() function.

This is based on an initial contribution discussed at the last SA3 meeting and is expected to reflect those discussions.

2 Issues

As specified in TS 29.198-3, the terminateAccess() function is digitally signed by the framework. To achieve this, two extra parameters are specified as input to the function: signingAlgorithm which identifies the algorithm used by the framework to produce the signature, and digitalSignature which contains the signature value itself.

2.1 Issue#1: no indication of public key/certificate to be used by verifier

The framework does not indicate which public key/certificate the client must use to verify the signature. The assumption to be made in the current specification is that the client and the framework have some a-priori agreement in which the client obtains a copy of the public key (embedded in a certificate or not) used by the framework for signing.

However, such a solution is not scalable and indication of the public key used is particularly important as the framework may have several (private/public) key pairs so that the client knows which one to use. The certificate itself is also important as a basis for signature verification (so as to validate the public key itself first).

2.2 Issue#2: no anti-replay protection

As currently specified, the signature is calculated solely over the terminationText string. Because such a string is more of a constant nature (same string is used on many occasions), no mechanism is defined to prevent re-use of a digital signature by a third-party.

2.3 Issue#3: no negotiation of signature algorithm

The signature algorithm used by the framework is not negotiated as it is a parameter of the terminateAccess(0 function itself. There is therefore no way for the client application to indicate which algorithm(s) it supports and it must consequently merely accept what it receives. If the signing algorithm is not supported, the client cannot verify the signature and an exception will be generated but the effect will most

probably be that the association with the framework will be considered closed by the client itself. If the latter is the case, the lack of a priori agreement also opens the door to denial of service: an attacker can issue a terminateAccess() to the client with a signature algorithm that it knows is not supported by the client. In such a scenario, the signature value does not have to be valid since the client will not try and verify it.

2.4 Issue#4: specification of signature algorithm

The list of signature algorithms is provided in table TpSigningAlgorithm, which lists P_MD5_RSA_512 and P_MD5_RSA_1024 as possible algorithms. Such a reference to the use of MD5 with RSA for signing is not sufficient to determine what the exact mechanism to implement is. Moreover, the use of MD5 as hashing algorithm and especially a modulus of 512 bits for RSA are not advisable and are deprecated.

3 Solution

With regards to issue #1, a solution could be add a new parameter to the terminateAccess function, carrying the public key identifier or its certificate. Another solution is to have the digitalSignature field itself carrying the certificate. This can be achieved by using an appropriate digital signature format such as the one defined in Cryptographic Message Syntax (RFC 2630). CMS indeed defines a data structure to carry a digital signature, the signed data and the signer's certificate.

With regards to issue #2, a fresh value must be generated by the framework for use as input into the signing algorithm. If adopting CMS to carry the signature, CMS already contains a field to contain the signing time. The signing time can be used by both parties to detect replayed signatures, under the condition that the verifier keeps track of the last verified value.

A separate contribution discusses a proposed mechanism for the negotiation of the signature algorithm (issue #3).

With regards to issue #4, the list of algorithms must be more precisely defined and can also be extended to other signing algorithms. Such a list of possible algorithms is given in IETF draft draft-ietf-pkix-ipki-pkalgs-05.txt, which itself refers to RFC 2437 specifying in detail RSA-based signing mechanism, to FIPS-186 for DSA signing mechanism and X9.62 for ECDSA signing mechanism.

The attached CR implements the corresponding changes to TS 29.198-3.

3GPP TSG CN WG5 **Document**

e.g. for 3GPP use the format TP-99xxx
or for SMG, use the format P-99-xxx

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-3** **CR** ⌘ ev **-** ⌘ Current version: **4.4.0** ⌘

*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 ☒

Title:	⌘	Security of terminateAccess() function
Source:	⌘	Alcatel
Work item code:	⌘	***Date:*** ⌘ 06-04-02
Category:	⌘	**F** ***Release:*** ⌘ Rel-4

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: ⌘

Issue#1: no indication of public key/certificate to be used by verifier

The framework does not indicate which public key/certificate the client must use to verify the signature. The assumption to be made in the current specification is that the client and the framework have some a-priori agreement in which the client obtains a copy of the public key (embedded in a certificate or not) used by the framework for signing.

However, such a solution is not scalable and indication of the public key used is particularly important as the framework may have several (private/public) key pairs so that the client knows which one to use. The certificate itself is also important as a basis for signature verification (so as to validate the public key itself first).

Issue#2: no anti-replay protection

As currently specified, the signature is calculated solely over the terminationText string. Because such a string is more of a constant nature (same string is used on many occasions), no mechanism is defined to prevent re-use of a digital signature by a third-party.

Issue#3: no negotiation of signature algorithm

The signature algorithm used by the framework is not negotiated as it is a parameter of the terminateAccess(0 function itself. There is therefore no way for the client application to indicate which algorithm(s) it supports and it must consequently merely accept what it receives. If the signing algorithm is not supported, the client cannot verify the signature and an exception will be generated but the effect will most probably be that the association with the framework will be considered closed by the client itself. If the latter is the case, the lack of a priori agreement also opens the door to denial of service: an attacker can issue a terminateAccess() to the client with a signature algorithm that it knows is not supported by the client. In such a scenario, the signature value does not have to be valid since the client will not try and verify it.

Issue#4: specification of signature algorithm

The list of signature algorithms is provided in table TpSigningAlgorithm, which lists

		P_MD5_RSA_512 and P_MD5_RSA_1024 as possible algorithms. Such a reference to the use of MD5 with RSA for signing is not sufficient to determine what the exact mechanism to implement is. Moreover, the use of MD5 as hashing algorithm and especially a modulus of 512 bits for RSA are not advisable and are deprecated.
Summary of change:	⌘	With regards to issue #1, the solution is to have the digitalSignature field carrying the certificate. This is achieved by using an appropriate digital signature format : the one defined in Cryptographic Message Syntax (RFC 2630). CMS indeed defines a data structure to carry a digital signature, the signed data and the signer's certificate. With regards to issue #2, a fresh value must be generated by the framework for use as input into the signing algorithm. CMS already contains a field to contain the signing time. The signing time can be used by both parties to detect replayed signatures, under the condition that the verifier keeps track of the last verified value. A separate contribution discusses a proposed mechanism for the negotiation of the signature algorithm (issue #3). With regards to issue #4, the list of algorithms is more precisely defined and can also be extended to other signing algorithms. Such a list of possible algorithms is given in IETF draft draft-ietf-pkix-ipki-pkalgs-05.txt, which itself refers to RFC 2437 specifying in detail RSA-based signing mechanism, to FIPS-186 for DSA signing mechanism and X9.62 for ECDSA signing mechanism.
Consequences if not approved:	⌘	Security weaknesses in terminateAccess() function and insufficient details for correct interoperable implementations.

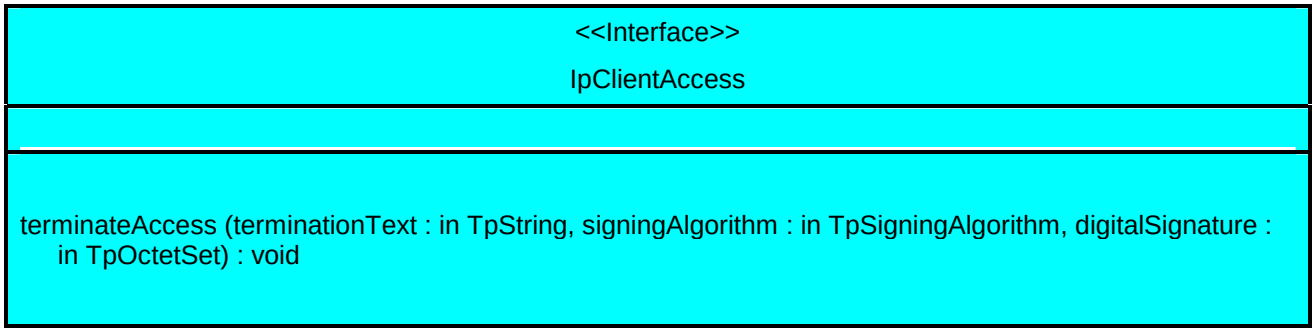
Clauses affected:	⌘	6.3.1.2, 10.3.11		
Other specs affected:	⌘	☐ Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	
Other comments:	⌘			

6.3.1.2

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.



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. 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. If the signingAlgorithm 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`

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~~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.
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 ~~key~~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.
P_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.

3GPP TSG CN WG5

Apr 8-12, 2002

Sophia Antipolis, FR

Source: Alcatel

Title: Use of one-way hash function for CHAP in OSA

Document for: Adoption

Agenda item: T.b.d.

1 Introduction

This contribution identifies an issue in TS 29.198-3 v4.4.0 with regards to the one-way hash function (MD5) to be used to realize CHAP-based authentication. This is based on an initial contribution discussed at the last SA3 meeting and is expected to reflect these discussions.

2 Issue

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.

As it currently stands, the text merely states that, when using CHAP for authentication, a CHAP mechanism as per IETF RFC 1994 is to be used. RFC 1994 describes on one hand the format of packets for exchanging the challenge and the response and one the other hand specifies the use of MD5 for CHAP, in which the input into the MD5 function (or any other one-way function for that purpose) is made of the concatenation of the Identifier, the shared secret and the challenge string.

2.1 Issue#1: use of RFC 1994 packet formats

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.

If RFC1994 must only be followed for the MD5 processing rule it provides, then it should be clearly specified in the TS.

2.2 Issue#2: weak use of one-way hash function

The mechanism described in RFC 1994, and hence inherited in OSA authentication, for calculating the input into the one-way hash function MD5 has since then (1996) been shown to present some weaknesses wrt the level of security. New constructions for one-way hash functions, such as HMAC, have since then been developed to cope with such

issues. The use of MD5 alone as described in RFC1994 is no longer safe. Alternatives based on HMAC (HMAC-MD5 or HMAC-SHA1) must be put in place for challenge-based authentication.

However, as it currently stands, P_OSA_AUTHENTICATION is only associated to the RFC 1994 CHAP mechanism. There is therefore no means to make use of another authentication mechanism in the context of P_OSA_AUTHENTICATION. A separate contribution discusses a proposed solution to enable the smooth negotiation of the authentication mechanism to be used between the client and the framework.

3 Solution

With regards to issue#1 above, it is suggested that the use of the packet format defined in RFC 1994 is clarified. In particular, the value to be used for the Name field of the Challenge and Response packets must be clarified.

With regards to issue#2, two new challenge-based authentication mechanisms are proposed: HMAC_MD5_96 and HMAC_SHA1_96. These are defined resp. in RFC 2403 and 2404. A separate contribution discusses a proposed mechanism to enable the definition of such new authentication schemes and their negotiation.

The attached proposed CR implements modifications to TS 29.198-3 in order to solve issue#1.

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-3** **CR** ⌘ ev **-** ⌘ Current version: **4.4.0** ⌘

*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 ☒

Title:	⌘	Use of one-way hash function for CHAP
Source:	⌘	Alcatel
Work item code:	⌘	***Date:*** ⌘ 06-04-02
Category:	⌘	**F** ***Release:*** ⌘ Rel-4

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.

As it currently stands, the text merely states that, when using CHAP for authentication, a CHAP mechanism as per IETF RFC 1994 is to be used. RFC 1994 describes on one hand the format of packets for exchanging the challenge and the response and one the other hand specifies the use of MD5 for CHAP, in which the input into the MD5 function (or any other one-way function for that purpose) is made of the concatenation of the Identifier, the shared secret and the challenge string.

Issue#1: use of RFC 1994 packet formats

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.

If RFC1994 must only be followed for the MD5 processing rule it provides, then it should be clearly specified in the TS.

Issue#2: weak use of one-way hash function

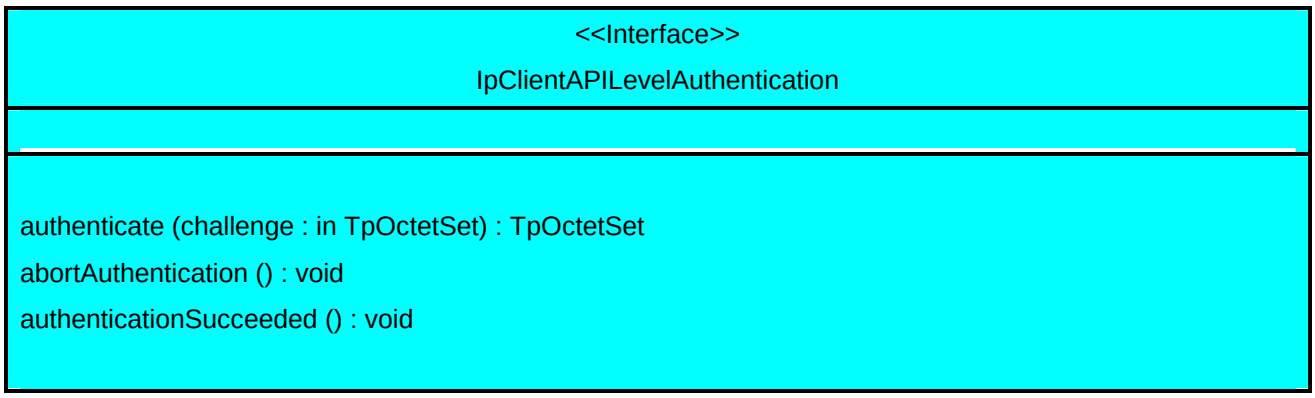
The mechanism described in RFC 1994, and hence inherited in OSA authentication, for calculating the input into the one-way hash function MD5 has since then (1996) been shown to present some weaknesses wrt the level of security. New constructions for one-way hash functions, such as HMAC, have since then been developed to cope with such issues. The use of MD5 alone as described in RFC1994 is no longer safe. Alternatives

		based on HMAC (HMAC-MD5 or HMAC-SHA1) must be put in place for challenge-based authentication. However, as it currently stands, P_OSA_AUTHENTICATION is only associated to the RFC 1994 CHAP mechanism. There is therefore no means to make use of another authentication mechanism in the context of P_OSA_AUTHENTICATION. A separate contribution discusses a proposed solution to enable the smooth negotiation of the authentication mechanism to be used between the client and the framework.
Summary of change:	⌘	With regards to issue#1 above, the actual use of the packet format defined in RFC 1994 is clarified. In particular, the value to be used for the Name field of the Challenge and Response packets are also be clarified. With regards to issue#2, two new challenge-based authentication mechanisms are proposed: HMAC_MD5_96 and HMAC_SHA1_96. These are defined resp. in RFC 2403 and 2404. A separate contribution discusses a proposed mechanism to enable the definition of such new authentication schemes and their negotiation.
Consequences if not approved:	⌘	Lack of detailed specification can lead to interoperability issues. Other hashing mechanisms must be provided to avoid potential security weaknesses in MD5 itself.

Clauses affected:	⌘	6.3.1.1, 6.3.1.5		
Other specs affected:	⌘	☐ Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	
Other comments:	⌘			

6.3.1.5 Interface Class IpClientAPILevelAuthentication

Inherits from: IpInterface.



Method

authenticate()

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().

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

Returns

TpOctetSet

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.

When an authentication algorithm different from MD5 has been negotiated, the algorithm that has been agreed upon shall be used to generate the response value.

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()`

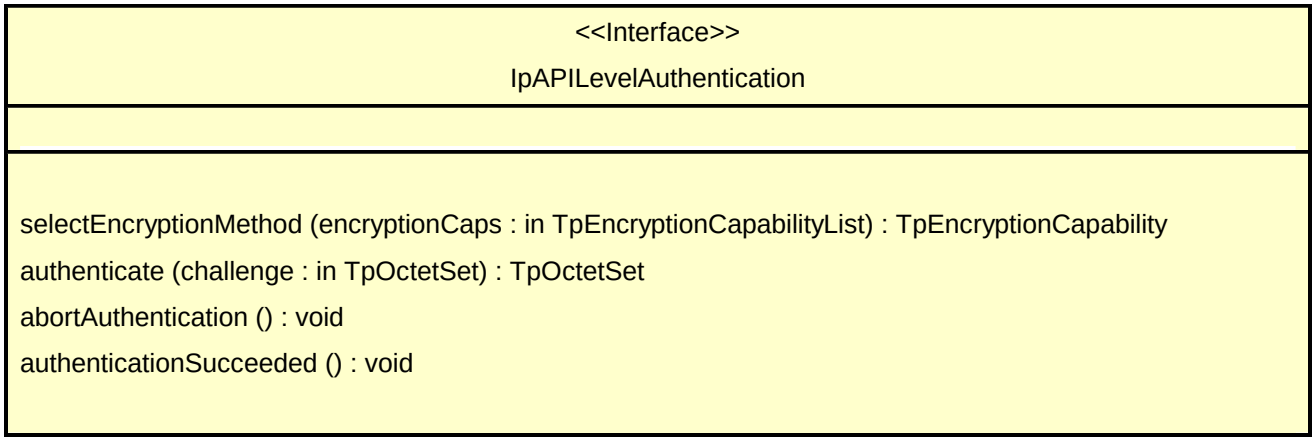
The Framework uses this method to inform the client of the success of the authentication attempt.

Parameters
No Parameters were identified for this method

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.



Method
`selectEncryptionMethod()`

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 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().

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

Returns

TpOctetSet

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.

When an authentication algorithm different from MD5 has been negotiated, the algorithm that has been agreed upon shall be used to generate the response value.

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