WP33 - eSign Services

D33.2 - Interface and Module Specification and Documentation

Document Identification	
Date	16.12.2013
Status	Final
Version	0.6

Related SP / WP	SP3 / WP33	**Document Reference**	D33.2
Related Deliverable(s)	D23.2, D23.3, D31.2, D32.3, D33.1	**Dissemination Level**	PU
Lead Participant	SK	**Lead Author**	Maili Keskel (SK)
Contributors	Moritz Horsch (TUD), Mohamed El Yousfi (TUD), Christof Rath (TUG), David Derler (TUG), Tobias Wich (ECS), Florian Gruner (AG), Alexander Gast (AG), Eray Özmü (USTUTT) Maili Keskel (SK)	**Reviewers**	Detlef Houdeau (IFAG), Miguel Colomer (ATOS)

This document is issued within the frame and for the purpose of the FutureID project. This project has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318424

Abstract

The present document provides a brief overview of interfaces of eSign service Module as such and also the interfaces of the underlying framework in FutureID infrastructure. This work and deliverable is embedded between the requirement analysis (D33.1), the implementation (33.3), the integration (D33.4), provision of the selected evaluation documents (D33.5) and the legal analysis of eSignature services (D33.6).

Document Information

History

Version	Date	Author	Changes
0.1	04.09.2013	Moritz Horsch	Created document structure
0.2	13.09.2013	Moritz Horsch	Added description of the FutureID infrastructure
0.3	07.10.2013	Christof Rath, David Derler	Added description of the eSign Service interface
0.4	30.10.2013	Florian Gruner, Alexander Gast	Added description of Trusted Viewer
0.5	12.12.2013	Moritz Horsch	Updated terms to new Reference Architecture
0.6	16.12.2013	Eray Özmü	Added summary of the requirements

Table of Contents

Table of Figures

1. Introduction

For starters this sections gives an overview about the document broadly.

1.1 Scope

The role of this WP: “eSign Services”, is the development of a framework that is capable to process digital signature related tasks, like signature creation and verification. The framework will be interfaced via the OASIS-DSS protocol. However, the design shall be flexible enough to be interfaced via different protocols and bindings.The various formats of advanced digital signatures will be connected to the framework as plugins allowing to select implementation of different vendors (FutureID-DOW) and to extend the service further formats.

1.2 Outline

This document is structured as follows: In the next section it shortly describes the FutureID infrastructure as well as the architecture and the components of the FutureID client. This is followed by the brief summary of the requirements for the eSign Services of the FutureID client. Thereafter we give a high level overview over the eSign Service interface of the FutureID client. The last section focuses on the Trusted Viewer.

1.3 Terminology

This section describes the meaning of the most important words used in this document to precise communication and to provide a common understanding.

1.3.1 Key Words

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

1.3.2 Abbreviations and Notations

This section itemizes the abbreviations and notations used in this document.

AIS	Application Integration Services
BS	Broker Service
CAdES	CMS Advanced Electronic Signature
CV	Credential Viewer
CMS	Cryptographic Message Syntax
CSP	Cryptographic Service Provider
DoW	Description of Work
DSA	Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
eSign	Electronic Signature
FC	FutureID Client
FS	Federation Service
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol

IdM	Identity management
IFD	Interface Device
JCA	Java Cryptography Architecture
OASIS	Advancing Open Standards for the Information Society
PAdES	PDF Advanced Electronic Signature
PDF	Portable Document Format
RFC	Request for Comments
SAL	Service Access Layer
SHA	Secure Hash Algorythm
SOAP	Simple Object Access Protocol
SP	Service Provider
TS	Trust Service
WYSIWYG	What You See Is What You Get
XAdES	XML Advanced Electronic Signature
XML	Extensible Markup Language
XSD	XML Schema Definition

1.3.3 Terms

This section provides descriptions for selected terms that are used in this document. The descriptions are taken from the FutureID terminology [2].

application
An Application is a service consumed by the user and integrated into the FutureID infrastructure via the Application Integration Services. The AIS may be a government service for citizens (ie. health service, public administration service) as well as business services which can be used by both large and small enterprises.

attribute
An attribute is a physical or abstract property belonging to an entity. An attribute typically consists of a name value pair.

attribute value
An attribute value is a particular instance of the class of information indicated by an attribute type component which indicates the class of information given by that attribute.

authentication
Authentication is the process to ensure that the individual is who he or she claims to be.

authentication protocol
An authentication protocol is a protocol used to confirm the identity of an entity. It usually consists of the presenting of an identifier (e.g. a 'user-ID') and aone of more authentication factors (e.g., the knowledge of a secret).

availability
The availability can be described as the property of being accessible and useable upon demand by an authorized entity. In the context of service level agreements, availability generally refers to the degree to which a system may suffer degradation or interruption in its service to the customer as a consequence of failures of one or more of its parts.

binding
A binding is an explicit established association, bonding, or tie.

claim
A claim is an statement made by one entity about itself or another entity that a relying party considers to be in doubt until it passes Claims Approval.

confidentiality
Confidentiality means keeping the content of information secret from all entities except those that are authorized to access it.

cryptographic protocol
A cryptographic protocol is a distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve a specific security objective.

data
Data is a carrier of information. This information is encoded in a specific physical representation, usually a sequence of symbols that have meaning and can be processed or produced by a computer.

database
A database is a data repository, in which a collection of information organized in such a way that a computer program can quickly select desired pieces of data. Traditional databases are organized by fields, records, and files.

digital signature
A digital signature is data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the authenticity and integrity of the data unit. See also electronic signature.

electronic signature
An electronic signature is data in electronic form which is attached to or logically associated to other electronic data and serves as a method of authentication. From a legal perspective, an electronic signature is not necessarily considered equivalent to a handwritten signature. When it meets a number of conditions, it can be put on par with a handwritten one.

entity
An entity is an item of interest, inside or outside a system, such as an automated process, a subsystem, a device, a person or group of persons that incorporates a specific set of attributes.

federated identity
A federated identity is a partial identity which is the result of federation, and which usually implies that the entity to which this identity corresponds is recognized by several service providers or applications that are part of the federation.

identity
The identity of an entity is the dynamic collection of all of the entity's attributes. As such, the identity of an entity this is more a fluid and evolving ("philosophical") concept, rather than a practical one. An entity has only one identity, but it is neither possible nor desirable for an

information system to gather all the attributes of a specific entity. Information systems focus on a specific subset of relevant attributes, commonly referred to as ’partial identities’.

identity management (IdM)
The definition, designation and administration of identity attributes and the management of access to and the usage of applications, services and resources.

integrity
Integrity implies that the items of interest (facts, data, attributes etc.) have not been subject to manipulation by unauthorized entities.

name
A name is the identifier of an entity (e.g., subscriber, network element) that may be resolved/translated into an address.

person
A person can be a natural person (a human being) or a legal person.

personal data
Personal data is any information relating to an identified or identifiable natural person (the data subject).

policy
A policy is one or more definite goals, courses or methods of action to guide and determine present and future decisions. Policies are implemented or executed within a particular context (such as policies defined within an organization or business unit) or across contexts (e.g., in the case of an identity federation). Common examples of such policies are security policies, privacy policies, access control policies, registration policies etc.

profile
A profile of an entity or a group of entities is an organized set of attributes that characterizes the specific properties of that entity or entities within a given context for a specific purpose.

protocol
A protocol may be described as a set of rules (i.e., formats and procedures) to implement and control some type of association (e.g., communication) between systems (e.g. an internet protocol). In more general terms, a protocol can be qualified a series of ordered steps involving computing and communication that are performed by two or more system entities to achieve a joint objective.

role
A role is a set of one or more authorisations related to a specific application or service.

sector
A sociological, economic, or political subdivision of society.

service
A service is a digital entity comprising software, hardware and / or communications channels that interacts with subjects.

service provider (SP)
This is an entity that provides services to other entities.

token
A token is any hardware or software component that contains credentials related to attributes. They may take any form, ranging from a digital data set to smart cards or mobile phones. They can be used for both data/entity authentication and authorization purposes and they can contain entity's attributes.

transparency
Transparency is a data protection/privacy goal that ensures that all privacy-relevant data processing including the legal, technical and organizational setting can be understood by all involved parties including the Users.

unlinkability
Unlinkability is a privacy goal that ensures that all data processing is operated in such a way that the privacy-relevant data cannot be linked across privacy domains or used for different purposes than initially intended.

validation
Validation means confirming that information given is correct, often by seeking independent corroboration or assurance.

verification
The process or an instance of establishing the truth or validity of something.

verifier
Entity that corroborates a claim with a specified or understood level of confidence.

2. FutureID Architecture

This section shortly describes the FutureID infrastructure as well as the architecture and the components of the FutureID client to provide a brief overview of the different participants and components used in FutureID.

2.1 Infrastructure

This section provides a brief overview of the architecture of the FutureID infrastructure. FutureID builds a comprehensive and ubiquitous identity infrastructure for Europe including existing eID technologies and building on well-established standards. The FutureID infrastructure is illustrated in Figure 1 and comprises the User Platform, Service Provider, and FutureID Infrastructure Services. In detail, the FutureID infrastructure includes a Broker Service (BS), Federation Service (FS), Trust Service (TS), and a Credential Verifier (CV). Please note that multiple instances of the Broker Service, Federation Service, and Credential Verifier may exist.

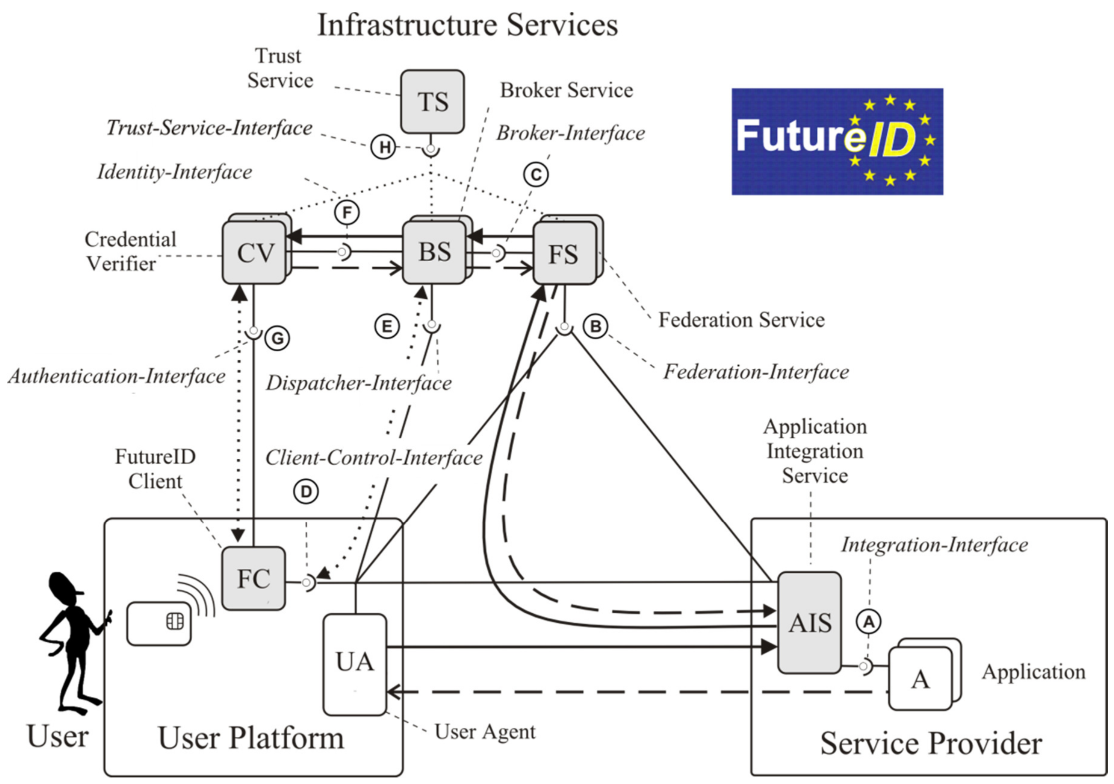

Figure 1: FutureID infrastructure

The Broker Service provides a common interface for service providers to connect to the FutureID infrastructure and hereby easily integrate arbitrary eID services using various credentials. The Broker Service provides two operation modes: First, in the Dispatcher Mode the Broker Service determines an appropriate existing credential verifier for the service provider, which is capable and suitable to authenticate the user. Second, in the Claims Transformer Mode the Broker Service includes the Universal Authentication Service as an internal credential verifier to perform user authentication and to issue session and attribute-based credentials. The Dispatcher Mode might cause message overhead and needs multiple redirects to enable the authentication of the user. To provide an optimised message flow and to enable the Claims Transformer Mode the Broker Service and the Universal Authentication Service takes care about the user authentication. Moreover, the Universal Authentication Service provides a novel approach to integrate arbitrary authentication protocols. The Universal Authentication Service operates a Job Execution Environment to run authentication protocols that are described in the Authentication Protocol Specification (APS) language, which allows describing authentication protocols in an abstract and convenient manner.

The Federation Service provides a federation-dialect-specific interface for service providers to get integrated into the FutureID infrastructure. It receives the authentication requests from the service provider's Application Integration Service and accesses the Broker Service through the Broker Interface. Please note that a Broker usually runs multiple Federation Services, in detail, one for each supported federation dialect like SAML (Security Assertion Markup Language).

The Credential Verifier verifies user credentials and issue session or attribute-based credentials. Such a Credential Verifier can be internal or external to the Broker Service. For instance, the Universal Authentication Service acts as an internal CV. The Trust Service is attached to the Broker Service and provides a comprehensive repository for trusted certificates and services as well as other trust related information (cf. WP43). Further the Trust Service is capable of evaluating certificates and assertions.

The Applications (A) operated by service providers encapsulate services that are integrated into the FutureID infrastructure using the Application Integration Service (AIS) and host services that are consumed by the users.

The FutureID Client (FC) is an eID application that allows end-users to use arbitrary credentials and authentication methods for services within the FutureID infrastructure. It is developed in SP3 and implements among other things the user interaction as well as the transport and security protocols. In detail, it includes the Interface Device Service (WP31), eID Services (WP32), eSign Services (WP33), User Interface (WP34), Trustworthy Client Platform (WP35), Browser Integration (WP36), and Client Testbed (WP37).

2.2 Client

This section provides a brief description of the individual components and the architecture of the FutureID client. The components are illustrated in Figure 2. Please note that this deliverable only considers the eSign Services. For additional information about the other components please see the respective work packages.

The Interface Device (IFD) is part of WP31 and specified in D31.2 [3]. The Event Manager, the Service Access Layer (SAL), and the Add-on Framework belong to this WP32. The signature components, as illustrated as the CAdES, XAdES, and PAdES add-ons in Figure 2, are part of the eSign Service in WP33 and described in this deliverable. The Graphical User Interface (GUI) is developed in WP34. WP35 covers the development of a trustworthy platform for computer and mobile devices for the FutureID client. Further, the Bindings component is developed in WP36.

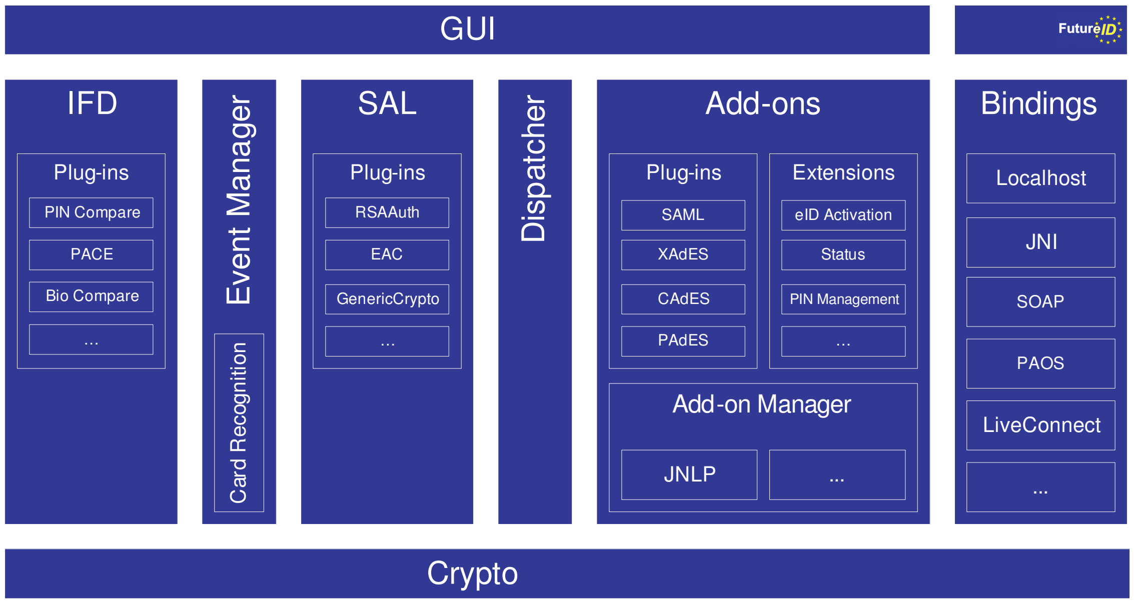

Figure 2: Architecture of the FutureID client

The individual components are described in the following:

2.2.1 Interface Device

The Interface Device (IFD) service provides a common interface for communication with arbitrary credentials. In detail, the IFD encapsulates card terminals, smart cards and secure elements and provides an interface to easily access such devices. Therefore, users, developers, and applications must not consider how such devices are integrated or addressed. The IFD abstracts from specific interfaces and physical properties like a contactless interface. For additional information please see D31.2 [3].

2.2.2 Event Manager

Users may connect or disconnect card terminals and insert or eject credentials at any time while the FutureID client is running. The objective of the Event Manager is to detect such events and to inform registered components about certain events, e.g., a new card terminal is detected or a card is removed. Further, such events should be used to provide user feedback, for instance, the graphical user interface shows a message that a new card is detected. The Event Manager also performs the recognition of card to determine the type and the functionality. The Event Manager is addressed in D32.3 [4].

2.2.3 Service Access Layer

The Service Access Layer (SAL) provides a generic interface for common credential services. This comprises connection, card application, data, identity, cryptographic, and authorization services. The SAL allows establishing a connection between the client application and a card application provided by a smart card or a credential, which is connected to the IFD. Card applications and card application services can be managed and executed by the SAL. The SAL also provides access to data that is stored on the card, so-called Data Structures for Interoperability (DSI), and allows creating, writing, reading, and deleting such data files and data sets. The Cryptographic Services and Authorization Services of the SAL provide common cryptographic functions for encryption and signature as well as allow managing access control rules for card applications. The SAL also allows to manage Differential Identities (DID) which describe objects for authentication and cryptographic usage, i.e., different cryptographic keys, e.g., for encryption and signature. The Differential Identity Services also provide an extension mechanism to support arbitrary authentication protocols. The expandability allows encapsulating arbitrary authentication technologies into the SAL without changing the interface or the implementation. The SAL is addressed in D32.3 [4].

2.2.4 Dispatcher

The Dispatcher provides a centralized communication component for handling incoming and outgoing messages. It manages a list of messages and the responsible component, i.e., the target component of a message. In detail, if the Dispatcher receives a message it determines the responsible component, for instance, the IFD or the SAL and forwards the message to the component. The Dispatcher is used, for instance, by the IFD, Event Manager, SAL, and the Bindings for internal and external message exchange. For additional information please see WP36.

2.2.5 Add-ons

Add-ons provide additional functionality and enhance the FutureID client. For instance, a PIN management and electronic signature functionality can be realized as an add-on and can be comfortably installed by the user. Further, supporting add-ons enable customization of the FutureID client and allow developers to easily extend the application. The Add-on Framework comprises the support of arbitrary protocols, credentials, and add-ons to enhance the functionality of the FutureID client. It is comprehensively described in D32.3 [4].

2.2.6 Bindings

The Bindings component comprises modules for exchanging data between different entities. The particular components implement a protocol like HTTP or SOAP to transmit messages between the FutureID client and external applications or external services. For example, a binding component will handle and transmit the messages between the FutureID client and the Identity Broker.

2.2.7 Graphical User Interface

The Graphical User Interface (GUI) component provides an abstract framework to develop user interfaces and user interactions. Other components can use this framework to define the user interaction like confirmation and information dialogs. Furthermore, the components are independent from an actual platform-specific GUI interface like Java Swing or the Android GUI. The respective rendering, i.e., the mapping of an abstract user interaction element (e.g., a button) to a respective GUI interface like Java Swing can be implemented in a platform-specific module.

2.2.8 Crypto

Many of the client components use cryptographic functions, e.g., for encrypted communication, message integrity, key exchange, etc. This comprises for example encryption (e.g., AES) and signature (e.g., DSA) schemes as well as hash functions (e.g., SHA-1). The Crypto component encapsulates common cryptographic functions that are used by other components. The cryptographic primitives will be provided by a Cryptographic Service Provider (CSP) like Bouncy Castle [5].

3. Requirements

This section provides a brief summary of the requirements for the eSign Services of the FutureID client. The requirements are condensed from the comprehensive analysis in D33.1 [6] and separated into functional and non-functional requirements. The following chapters 3.1 and 3.2 only represent a subset of the compiled requirements. Therefore it is highly recommended for developers to also read through the deliverable D33.1.

3.1 Non-Functional Requirements

The eSign Services MUST comply with the reference architecture and the add-on framework of the FutureID client (Req. 5.1, Req. 5.18)

The interfaces of the eSign Services SHOULD be easy to test and the interfaces should be accessed via different operating systems. Furthermore the eSign Services MUST be documented properly and allow further major version upgrades.

Legal and licensing issues MUST be avoided. Country specific legal necessities MUST be implemented and automatically updated.

3.2 Functional Requirements

The eSign Services module MUST support different signature formats namely CAdES, XAdES and PAdES. These formats MUST be supported with the –BES, -LT and –LTA levels.

The generators for the signature formats XAdES and CAdES MUST furthermore be able to bind the source data in a single file. This can be done by using the ASiC container, which MUST conform to the ASiC standard and its Baseline Profile.

To enable remote signing and signature verification the eSign service MUST provide an interface of the OASIS-DSS protocol.

The eSign Services MUST enforce appropriate access rules – only access for authorized users.

If FutureID infrastructure provides the function to sign electronically it MUST adhere to the applicable national restrictions/requirements for electronic signatures (Req. 5.13)

Restricted electronic certificates MUST be used within the borders of their restrictions, also for restrictions by the terms&conditions of the certificate service provider and for restrictions by member states (Req. 5.14, Req. 5.15, Req. 5.16)

4. Interface specification and documentation

This section gives a high level overview over the eSign Service interface of the FutureID client, which will be discussed in a more detailed way in the following sections.

Basically, the eSign Service of the FutureID client consists of one or more `SignaturePlugin` implementations which comply with the Add-on Framework specification in D32.3 [4], i.e., the `SignaturePlugin` implements the `ApplicationPluginAction` interface, which enables the communication with other components over bindings. Furthermore, the various supported signature formats, like XAdES, CAdES, and so forth, constitute single components, being usable by `SignaturePlugin` implementations. This way, supporting additional signature formats as well as adding `SignaturePlugin` implementations is possible.

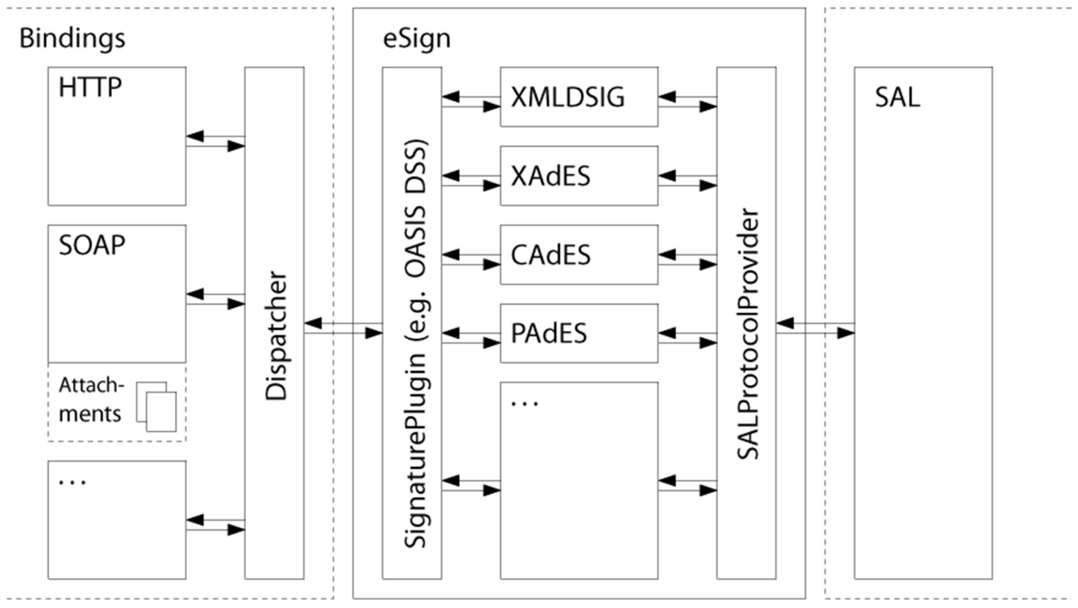

Figure 3: Componentized Overview over the eSign functionality of the FutureID client (adapted from D32.3)

Since the client most likely makes use of credentials, e.g., smart cards, for issuing digital signatures, a way for making the Service Access Layer (SAL) available to the libraries, being potentially used for signature creation and validation, is required. However, modifying those libraries is not an option, and, thus, we propose to make the SAL available over a Java Cryptographic Provider which provides the implementations making use of the required SAL protocols over the conventional Java Cryptography Architecture (JCA) provided factories, e.g., a `SignatureSpi` implementation wrapping a SAL protocol for signature creation on a smart card.

4.1 SignaturePlugin

Basically, the eSign Service of the FutureID client is required to support the OASIS DSS standard [7], whilst the design should allow for providing implementations of arbitrary other frameworks for signature creation and validation. Since we propose to build the signature service component upon the Add-on Framework of the FutureID client this design goal can be easily achieved. However, for the rest of this document we only consider an implementation of OASIS DSS together with various signature formats.

The OASIS DSS standard basically provides means for issuing and verifying signatures based on standardized requests and responses, which can either be encapsulated within HTTP or SOAP. Whilst OASIS DSS does not define how to handle attachments in the former case, MIME attachments, which can be referenced from the message body, can be included in the latter case. Obviously, the response of the service is again HTTP or SOAP, but without attachments in either case. Since the response contains a `dss:SignatureObject` the aforementioned attachment restriction does not prevent from returning arbitrary signatures formats, and is thus suited for our purposes. From an implementation point of view, the `execute(…)` method of the `ApplicationPluginAction` interface defined in D32.3 [4] exactly resembles the structure discussed above, and, thus, we propose to implement all signature services as `ApplicationPluginAction` (see Figure 4). This way, it is also possible to use protocols other than HTTP or SOAP for performing an OASIS DSS request, and, further, it is possible to easily implement other signature services by simply implementing another plugin.

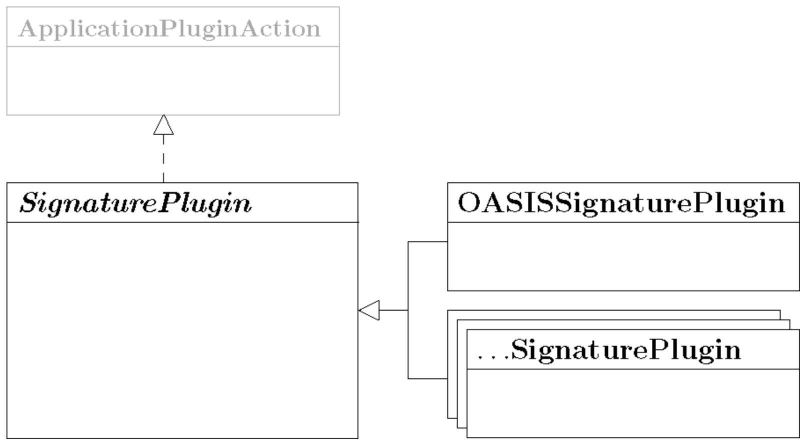

Figure 4: Signature Service Plugin

Since the eSign functionality of the FutureID client is also required to support multiple signature formats, i.e., XAdES, PAdES and CAdES, we propose to encapsulate the signature formats by using the `SignatureFormat` interface shown in Figure 5. In order to be able to instantiate the individual signature format implementations in a straight-forward manner, the abstract `SignatureFormat` class provides a static factory method.

The parameterization of the instantiations, thereby, is done by making use of the transparent `SignatureFormatParameters` interface. Furthermore, the `SignatureFormat` implementation being currently active can be set in the `SignatureFormatRegistry`, where it is also possible to register all available `SignatureFormat` implementations.

Moreover, we propose the following workflow for dispatching a sign or verify request. The `SignaturePlugin` dispatches the incoming request and hands on the relevant data, i.e., Documents (`Document`), optional input parameters (`OptionalInput`) and (in case of verify) the `SignatureObject` to the currently active signature format. The signature format then performs the requested operation on the documents and returns the result. For the `verify` method this return value is a `ValidationReport` containing the result conforming to ETSI TS 102 853 [8], whereas for the `sign` method this return value is a `SignatureObject` conforming to the OASIS DSS specification [7].

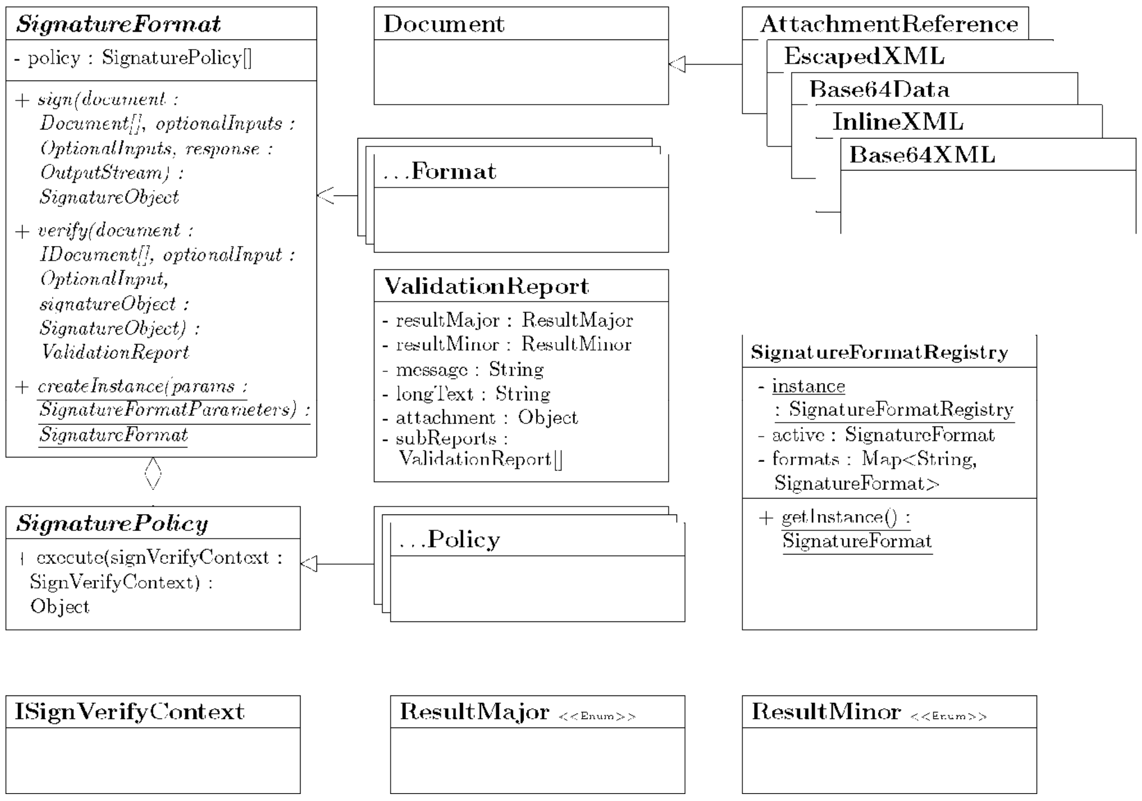
SignatureFormat
- policy : SignaturePolicy[]
+ sign(document : Document[], optionalInputs : OptionalInputs, response : OutputStream) : SignatureObject
+ verify(document : IDocument[], optionalInput : OptionalInput, signatureObject : SignatureObject) : ValidationReport
+ createInstance(params : SignatureFormatParameters) : SignatureFormat
Document
AttachmentReference
EscapedXML
Base64Data
InlineXML
Base64XML
...Format
ValidationReport
- resultMajor : ResultMajor
- resultMinor : ResultMinor
- message : String
- longText : String
- attachment : Object
- subReports : ValidationReport[]
SignatureFormatRegistry
- instance : SignatureFormatRegistry
- active : SignatureFormat
- formats : Map<String, SignatureFormat>
+ getInstance() : SignatureFormat
SignaturePolicy
+ execute(signVerifyContext : SignVerifyContext) : Object
...Policy
ISignVerifyContext
ResultMajor <<Enum>>
ResultMinor <<Enum>>

Figure 5: Signature Format Framework

4.1.1 Further Details on SignatureFormat Inheritances

`SignatureFormat` inheritances are intended to perform the sign and the verification operation for the respective signature format. Since most implementations will make use of libraries for signing and verifying signatures, we need to take a closer look at the potential architecture of such libraries and how it is possible to force those libraries to use IFD and SAL protocols, respectively, without the modification of the libraries (see Section 4.1.1.1).

Furthermore, each `SignatureFormat` inheritance contains a set of `SignaturePolicy` objects, enforcing or checking certain signature properties. Thus, each `SignaturePolicy` has access to an implementation of `SignVerifyContext`, which is a transparent interface allowing `SignatureFormat` implementations to pass the data being necessary for enforcing policies.

4.1.1.1 Facilitating the use of arbitrary signing credentials

As already mentioned before, a way for forcing libraries to use custom ways of accessing the signing and verification credentials, e.g., creating a signature on a smart card, is required. We propose to enable this in the following way:

Virtually all state of the art Java crypto libraries make use of the Java Cryptography Architecture (JCA), and, thus, also libraries implementing signature formats such as XAdES, internally retrieve the required algorithm implementations, e.g., Elliptic Curve Integrated Encryption Scheme (ECIES), using JCA provided means, which, in turn, can be used for forcing libraries to use implementations by a custom provider. In our case this could be a provider, providing implementations which interact with a SAL protocol. To do so, we propose to make use of a custom provider implementation that it is capable of supporting multiple SAL protocols by supplying the desired protocol over a `SALKey`, which implements both, the `PublicKey` and the `PrivateKey` interface, and is intended to pass the desired SAL protocol to the respective algorithm implementation (see Figure 6). By extending this interface, one can additionally pass arbitrary parameters.

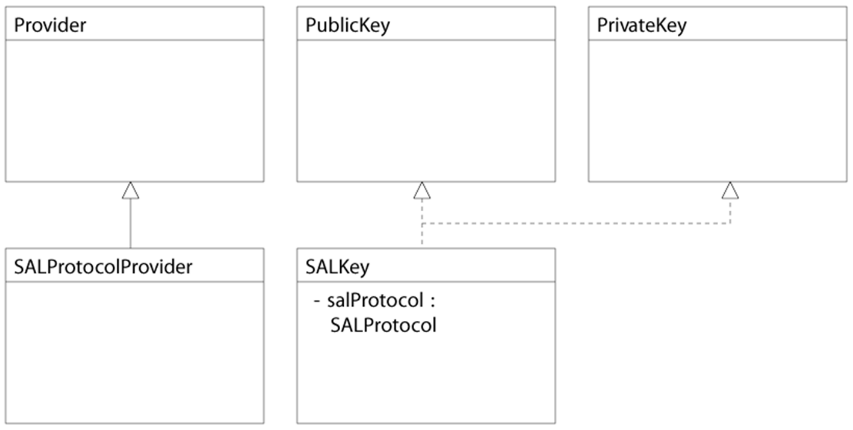

Figure 6: Provider for accessing SAL-protocols

Based on the provider defined above, one can, e.g., create a `SignatureSpi` implementation making use of the `salProtocol` field being held in the `SALKey` which is passed to the implementation using the `initSign` or `verify` method.

Figure 7 illustrates the retrieval of a Signature implementation from a `SALProtocolProvider`. Note that it is crucial to register the desired provider at the first position in order to guarantee that the implementations are taken from this provider (if available).

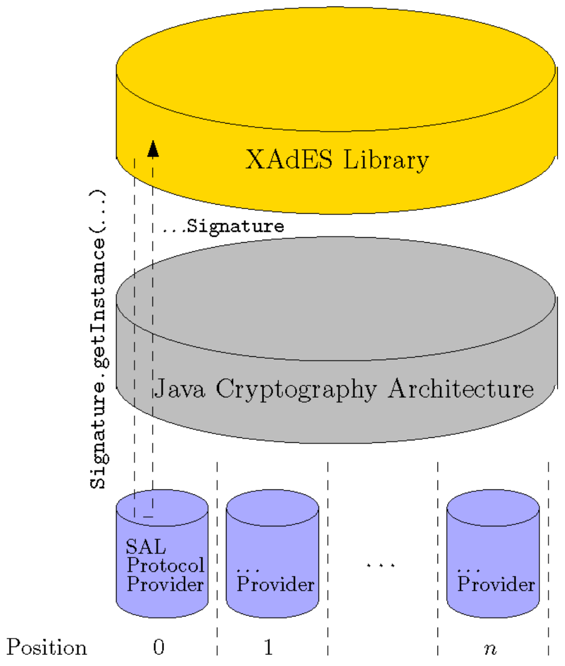

Figure 7: XAdES library retrieving Signature implementation from SALProtocolProvider (adapted from [9])

4.2 Open Issues and Potential Solutions

In this section we shortly mention identified issues of the current design or of particular parts of the eSign functionality of the FutureID client.

4.2.1 File System References in XAdES Signatures

The XAdES signature format allows for referencing files which are located on the file system within a signature. This, however, does not make sense, and could even be a threat from a security point of view for signature services running on a webserver. Thus, we propose to switch off the support for file system references by default and require to explicitly enabling the support for clients running localhost.

4.2.2 SignatureFormat Creation

In the current design, the used `SignatureFormat` is centrally set in the `SignatureFormatRegistry`, which prevents the use of multiple signature formats at the same time. This arises from the fact that using the Add-on Framework means that one never has access to the concrete classes implementing the add-on and thus it is not possible to keep the active signature format per add-on.

5. Trusted Viewer

5.1 Motivation

Sharing documents online is getting a convenient way for enterprises, government agencies and private users to provide partners and involved parties with information or even to establish contractual agreements. This changing practice becomes nearly universal and in some cases mandatory by law (see MADSig: Enhancing Digital Signature to Capture Secure Document Processing Requirements). In this way eContracting offers a couple of advantages for involved partners, such as automatic processing within different applications or even an automatic examination for completeness of content. So it would be possible to implement a monitoring system for contract fulfillment or to simplify the internal version control (see eContracting in "eDemocracy and eGovernment"). On the hand, many issues related to authentication of business partners, coding digital documents and utilization of digital signatures have been solved. But on the other hand, we still have to face and handle a major difficulty by using digital documents for business processes. Information of digital documents are represented by logical bits and bytes, which can be stored on, copied to any suitable electronic medium and they only become meaningful to humans when represented through an analogue physical medium such as a screen or a printout (see Wikipedia – WYSIWYG). Taking these properties of our digital environment into account, a possibility for fraud is offered.

5.2 Technical properties

A trusted viewer, or secure viewer, is intended to secure electronic business processes between participants. It provides a technical environment for trading digital documents, where rights and interests of involved parties are protected. For the document signer a trusted viewer is an appropriate measure to guarantee a digital trading process, which is free of data manipulations, identity fraud or crime or even man-in-the-middle attacks. To ensure such secure digital processes a trusted viewer has to match a basic technical requirement – it has to make sure, that the signer of a digital document is able to see the content, he wants to sign. In the literature this property is called "what you see is what you sign" or even shorter WYSIWYS (What You See Is What You Get). Therefore a trusted viewer requires a combination and interplay of two important components to provide users with a secure interface when using security critical applications. First of all it needs a trusted path between the user and the signature application, where no other application can access in- and output of this application. Besides that, a trusted viewer has to prevent the rendering component from being manipulated, which is realized by a trusted GUI (see http://www.trust.rub.de/projects/trustedviewer/).

A trusted path, respectively trusted channel, is a security architecture concept providing confidence for users and their communication with partners. By the utilization of such a mechanism, both participants can be sure, that their opposite is reliable. Furthermore attackers are prevented to intercept or modify transported information. After technical mechanism guaranteed a secure transportation of information a further utility has to proof data integrity. Before a rendering component shows digital content, the content must be verified. That means, information transported via a XML-standard have to be checked against a scheme, like XSD to ensure, that there is no secret information. Such hidden information in digital documents could be used to the disadvantage of the signer.

5.3 Implementation details

Currently, a basic decision related to the implementation details of a trusted viewer must be made – should it be either offered as a plug in (for example as an extension of the functionalities) within the FutureID-client environment or is it better to implement it as an standard functionality of the standard core right from the beginning. In such a security sensitive area of our digital world, users could be disconcerted about security questions, due to the reliability of the source of the trusted viewer plugin. Additionally, any client has to offer interfaces for the integration of a trusted viewer, which could be seen as a potential target for hackers. In this case, transported information could be manipulated without the knowledge of involved parties.

Advantages of a trusted viewer as a part of the standard core could be seen in the higher degree of user acceptance. If there is a separate development of a FutureID-client and a trusted viewer extension, users would be responsible to make sure, that they use the latest version of both applications. Furthermore such an integrated environment would be more reliable, stable and secure. From a user’s perspective and security reasons, an integrated and closed system could be more trustworthy.

6. Bibliography

[1] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels,* RFC 2119, 1997.

[2] FutureID, *WP21 - Vision, Approach and Inventory, D21.1 - FutureID Terminology,* 2013.

[3] FutureID, *WP31 - Interface Device Service, D31.2 - Interface and module specification and documentation,* 2013.

[4] FutureID, *WP32 - eID Services, D32.3 - Interface and Module Specification for eID Services,* 2013.

[5] Legion of the Bouncy Castle, *Bouncy Castle,* http://www.bouncycastle.org.

[6] FutureID, *WP33 - eSign Services, D33.1 - Requirements Report,* 2013.

[7] OASIS, *Digital Signature Service Core Protocols, Elements, and Bindings,* Version 1.0, OASIS Standard, http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf .

[8] European Telecommunication Standards Institute (ETSI), *Electronic Signature and Infrastructures (ESI); Signature verification procedures and policies,* ETSI TS 102 853.

[9] Oracle, *Java Cryptography Architecture (JCA) Reference Guide,* http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html .

[10] ISO/IEC, *Identification cards - Integrated circuit card programming interfaces - Part 3: Application interface,* International Standard, ISO/IEC 24727-3, 2008.

[11] Bundesamt für Sicherheit in der Informationstechnik (BSI), *eCard-API-Framework - ISO 24727-3-Interface,* Technical Guideline TR-03112-4, Version 1.1.2, 2012.

[12] Bundesamt für Sicherheit in der Informationstechnik (BSI), *eCard-API-Framework - Protocols,* Technical Guideline TR-03112-7, Version 1.1.2, 2012.

[13] FutureID, *WP32 - eID Services, D32.2 - Requirements report,* 2013.

[14] European Committee for Standardization (CEN), Identification card systems - European Citizen Card, CEN/TS 15480, Part 1 - 4, 2008.