WP33 - eSign Services

D33.3 – Implementation of the Framework

Document Identification	
Date	September 15, 2014
Status	Final
Version	1.02

Related SP/WP	SP3/WP33	**Document Reference**	http://is.gd/MDzBq2
Related Deliverable(s)	D23.2, D33.2	**Dissemination Level**	PU
Lead Participant	TUG	**Lead Author**	David Derler (TUG)
Contributors	David Derler (TUG) Christof Rath (TUG) Tino Liebeskind (AG) Christopher Ruff (USTUTT)	**Reviewers**	ULD, SK

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.

1 Executive Summary

In this document the implementation related aspects of the eSign Services are discussed. The design decisions as well as the implementation aspects, are, thereby, in line with D33.1 [25] and D33.2 [26], respectively. The eSign Services are implemented as an Add-on, conforming to the Add-on framework of the *FutureID* client and can, thus, be interfaced via the bindings framework. The main idea behind this concept is that the eSign Services operate as a localhost signing server, providing means for an OASIS DSS [38] based transparent signature creation, supporting a large variety of signature creation devices.

Thereby, incoming OASIS DSS requests, are dispatched and, depending on their type, i.e., `SignRequest` or `VerifyRequest`, handed on to the so-called `SigningServiceProxy` or to the so-called `VerificationServiceProxy`, respectively. These proxies allow for delegating the calls to arbitrary service implementations, and, thus, one is able to provide alternative ways for signature creation and verification when the default services, provided along with the *FutureID* client, do not suit the needs of a specific application. The `SigningService` implementation provided by default is capable of creating Advanced Electronic Signatures [9] in XML [2, 15], PDF [3, 12, 13] and CMS [4, 11] format. The verification of such a signature involves the validation of one or more certificates, linking a public verification key to a real-world entity. However, we assume that an average user is not capable of making decisions regarding the trust status of certain certificates and certification authorities, which is crucial for an adequate signature verification. Thus, we decided on outsourcing the default validation service to a web-service, which is provided by the Trust Services [28] component of the *FutureID* architecture, and, consequently, we assume the signature validation to be a black-box returning OASIS DSS `VerifyResponses` for `VerifyRequests` for the rest of this document. We further note that the aforementioned validation service is currently deployed at `https://trustservices.iaik.tugraz.at/trustServicesWeb/validation.wsdl`, and is documented in [29]. In contrast, the signature creation is performed directly at the client side.

From an OASIS DSS point of view, the OASIS DSS core [38] profile extended by *FutureID*-specific structures is supported. This is achieved by providing a *FutureID* profile, which extends the core profile with the required fields. These fields are, e.g, an optional input for signing/verification policies [1], an optional input for requesting an extension of a signature to a format being suited for long term validation (LT, LTA), and an optional input for requesting to wrap the signature in an ASiC signature container [16].

Finally, user interface components for performing signing/verification operations directly from the client GUI and a trusted viewer for viewing XML and PDF signatures before signing is provided.

2 Document Information

2.1 Contributors

Name	Affiliation
David Derler	TUG
Christof Rath	TUG
Tino Liebeskind	AG
Christopher Ruff	USTUTT

2.2 History

0.01	2014-03-07	David Derler	1st draft, document structure
0.02	2014-03-10	David Derler	Added introduction and eSign Services architecture section
0.10	2014-03-11	David Derler	Added signature format and signature policy section
0.20	2014-07-04	Tino Liebeskind	Added first draft of ASiC and trusted viewer section
0.21	2014-07-10	Tino Liebeskind	Some rework of ASiC and trusted viewer section
0.30	2014-07-15	Christopher Ruff	Added first draft of GUI section
0.32	2014-07-16	Christopher Ruff	Some rework and English text in figures
0.40	2014-07-17	David Derler	Added outline
0.41	2014-07-18	Christopher Ruff	Minor rework
0.50	2014-07-21	Tino Liebeskind	Trusted viewer section rework
1.00	2014-07-25	David Derler	Final consistency check and minor corrections
1.01	2014-08-11	David Derler	Incorporate review comments from SK
1.02	2014-09-05	David Derler	Incorporate review comments from ULD

2.3 Table of Contents

2.4 List of Figures

2.5 List of Tables

2.6 List of Acronyms

pki	public key infrastructure
idm	identity management

2.7 Glossary of Terms

access control
Prevention and protection of resources against unauthorised access; a process by which use of resources is regulated according to a security policy and is permitted by only authorised people according to that policy. Access control can be logical e.g. for IT systems, or physical e.g. for entry to buildings.

account
Typically a formal business agreement for providing regular transactions and services between a principal and the business service providers.

anonymity
The quality or state of being anonymous, which is the condition of having a name or identity that is unknown or concealed.

applicant
An individual applying for one or more services.

assertion
A set of statements that can be evaluated by an authority (usually an Identity Provider) concerning a principal. The statements can be concerning identifying information (e.g. name) as well as attributes (e.g. role). An assertion can also be thought of as being a set of claims about a principal.

attribute
Information bound to an entity that specifies a characteristic of that entity, such as a group membership or a role, or other information associated with that entity. There are a number of different types of Attribute. Absolute Attribute, Distinguished Attribute and Biographical Attribute.

attribute value
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
The corroboration of a claimed information (e.g. a set of attributes) with a specified, or understood, level of confidence.

Authentication may be used during any identity management process. Authentication serves to demonstrate the integrity (i.e. equivalence to a corresponding reality) and origin (i.e. the source) of what is being pretended (the claimed information).

The security and reliability of authentication mechanisms may vary dependant on the desired authentication level. The stronger the authentication, the higher the confidence that an entity corresponds with the claimed set of attributes.

Authentication is typically subdivided into two separate classes: data authentication and entity authentication. For this reason, autonomous use of the term authentication (without specifying the term of authentication) should be avoided, as it is subject to (mis)interpretation.

Authentication can be unilateral or mutual. Unilateral authentication provides assurance of the identity of only one entity, where mutual authentication provides assurance of the identities of both entities.

characteristic
A characteristic of an entity is an attribute specific to a particular context. A characteristic does not need to uniquely identify an entity but indicates an entity's capacity, function, qualification, etc.

claim
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
The avoidance of disclosure of information without the permission of its owner.

consent
The general permission granted by an individual to a requesting entity to use the individual's personal information in some agreed manner.

Consent can be expressed, implied, or provided through an authorized representative.

context
A sphere of activity, a geographic region, a communication platform, an application, a logical or physical domain, or alike that contribute to fix the environment in which an entity exists and is recognized.

credential
Data issued to an individual by a third party with a relevant authority or assumed competence to do so that is presented to provide evidence of a claim. A credential is a piece of information asserting to the integrity of certain stated facts.

delegation
Process in which an identified entity issues a mandate to another identified entity.

eID services
Services for entity authentication and signing data.

electronic identity
An electronic identity is a collection of identity attributes in an electronic form (eq. digital identity).

entity
An entity can be a physical or legal person, an animal, an organization, an active or passive object, a device, or a group of these individuals or anything that shall be characterised through the measurement of its attributes in an eIDM system. An entity is something that has separate and distinct existence and that can be uniquely identified.

federated identity
Credential of an entity that links an entity's partial identity from one context to a partial identity from another context.

identifier
An attribute or a set of attributes of an entity which uniquely identifies the entity within a certain context. Examples may include name, national insurance number, certificate numbers, encoded fingerprint, etc.

identity
The fact of being what an entity (person or a thing) is, and the characteristics determining this. It is a collection of attributes.

identity claim
A statement of something which pertains to the identity of an entity (e.g. a person's identity). An identity claim could convey a single attribute such as an identifier (e.g., a student number) or it could convey that a person is part of a certain group or has certain entitlements (e.g., "I am over 18", "I am a company employee"). A set of identity claims could provide sufficient identity attributes (e.g., name, date of birth, address) to permit the identification of a unique "identity" or person.

identity management
A unified framework for managing entities information and access rights across multiple systems and business contexts and linking individuals to the established identity when required. It enables convergence of the multitude of points of authentication, authorisation and administration to provide a more coherent view and management platform.

person
An entity recognized by the legal system. In the context of eID, a person who can be digitally identified.

principal
A principal is synonymous with an Identifiable Entity. A principal could be any identifiable entity, e.g a computer system, but it usually refers to an identifiable user of a system.

privacy
Ability of an individual to control access to personal information about themselves and the right to control the collection, storage, and dissemination of that information. It includes the right to see what personal information is stored and to correct or remove the access to that information as far as the law permits.

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.

providers
A generic way to refer to both Identity Providers and Service Providers.

public key infrastructure
A set of systems for managing paired public and private keys which Principals (including users and systems) can use to authenticate to each other, to sign documents and to send private messages to each other. It is a type of "trust infrastructure".

qualified electronic signature
Advanced electronic signatures which are based on a qualified certificate and which are created by a secure-signature-creation device, as defined in the eSignatures Directive.

registration
The process in which the entity is identified and/or other attributes are corroborated. Because of the registration, a partial identity is assigned to the entity for a certain context.

relying party
An individual, organization or service that depends on claims provider about a subject to control access to and personalization of a service.

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

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

service provider
A role donned by a system entity where the system entity provides services to Principals or other system entities. This would usually refer to a web site or other application provider.

token
Any hardware or software that contains credentials related to attributes. Tokens may take any form, ranging from a digital data set to smart cards or mobile phones. Tokens can be used for both data/entity authentication (authentication tokens) and authorisation purposes (authorisation tokens). Tokens can be separated into "Hardware Tokens" and "Software Tokens".

trust
Trust is the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context

trust infrastructure
The technical infrastructure used by system entities in order to trust each other. Trust infrastructures may be built on public key infrastructure, Kerberos, etc.

validation
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.

2.8 List of References

[1] ETSI TR 102 038. XML format for signature policies, v1.1.1, Apr 2002. URL http://docbox.etsi.org/EC_Files/EC_Files/tr_102038v010101p.pdf.

[2] ETSI TS 103 171. XAdES Baseline Profile, v2.1.1, Mar 2012, Mar 2012. URL http://www.etsi.org/deliver/etsi_ts/103100_103199/103171/02.01.01_60/ts_103171v020101p.pdf.

[3] ETSI TS 103 172. PAdES Baseline Profile, v2.2.2, Apr 2013. URL http://www.etsi.org/deliver/etsi_ts/103100_103199/103172/02.02.02_60/ts_103172v020202p.pdf.

[4] ETSI TS 103 173. CAdES Baseline Profile, v1.1.1, Sep 2011. URL http://www.etsi.org/deliver/etsi_ts/103100_103199/103173/01.01.01_60/ts_103173v010101p.pdf.

[5] ETSI TS 103 174. ASiC Baseline Profile, v2.2.1, Jun 2013. URL http://www.etsi.org/deliver/etsi_ts/103100_103199/103174/02.02.01_60/ts_103174v020201p.pdf.

[6] ISO 19005-1:2005. Document management — Electronic document file format for long-term preservation — Part 1: Use of PDF 1.4 (PDF/A-1).

[7] ISO 19005-2:2011. Document management — Electronic document file format for long-term preservation — Part 2: Use of ISO 32000-1 (PDF/A-2).

[8] ISO 19005-3:2012. Document management — Electronic document file format for long-term preservation — Part 3: Use of ISO 32000-1 with support for embedded files (PDF/A-3).

[9] EC 1999/93. Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures, Dec 1999. URL http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31999L0093:EN:NOT.

[10] ETSI TR 102 272. ASN.1 format for signature policies, v1.1.1, Dec 2003. URL http://www.etsi.org/deliver/etsi_tr/102200_102299/102272/01.01.01_60/tr_102272v010101p.pdf.

[11] ETSI TS 101 733. CMS Advanced Electronic Signatures (CAdES), v1.8.3, Jan 2011. URL http://www.cryptopro.ru/sites/default/files/products/tsp/ts_101733v010803p.pdf.

[12] ETSI TS 102 778-3. PDF Advanced Electronic Signature Profiles; Part 3: PAdES Enhanced - PAdES-BES and PAdES-EPES Profiles, v1.2.1, Jul 2010. URL http://www.etsi.org/deliver/etsi_ts/102700_102799/10277803/01.02.01_60/ts_10277803v010201p.pdf.

[13] ETSI TS 102 778-4. PDF Advanced Electronic Signature Profiles; Part 4: PAdES Long Term - PAdES-LTV Profile, v1.1.2, Dec 2009. URL http://www.etsi.org/deliver/etsi_ts/102700_102799/10277804/01.01.02_60/ts_10277804v010102p.pdf.

[14] ETSI TS 102 853. Signature verification procedures and policies, v1.1.1, Jul 2012. URL http://www.etsi.org/deliver/etsi_ts/102800_102899/102853/01.01.01_60/ts_102853v010101p.pdf.

[15] ETSI TS 101 903. XML Advanced Electronic Signatures (XAdES), v1.4.2, Dec 2010. URL http://uri.etsi.org/01903/v1.4.1/ts_101903v010401p.pdf.

[16] ETSI TS 102 918. Associated Signature Containers (ASiC), v1.2.1, Feb 2012. URL http://www.etsi.org/deliver/etsi_ts/102900_102999/102918/01.02.01_60/ts_102918v010201p.pdf.

[17] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP), Aug 2001. URL https://tools.ietf.org/html/rfc3161.

[18] D. Bratko, C. Hanser, and B. Haas. IAIK-JCE 5.1 SDK, 2013.

[19] D. Cooper, S. Santesson, S. Farrell, S. Boyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, May 2008. URL https://tools.ietf.org/html/rfc5280.

[20] D. Derler. On the Optimization of two Recent Proxy-Type Digital Signature Schemes and their Efficient Implementation in Java. Master's thesis, Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, 2013.

[21] D. Derler, C. Rath, M. Horsch, and T. Wich. Design und Implementierung eines Localhost Signaturgateways. In *Proceedings D-A-CH Security*, 2014.

[22] FutureID. WP31 - Interface Device Service, D31.2 - Interface and Module Specification and Documentation, 2013.

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

[24] FutureID. WP32 - eID Services, D32.4 - Implementation of Basic and Generic Modules, 2013.

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

[26] FutureID. WP33 - eSign Services, D33.2 - Interface and Module Specification and Documentation, 2013.

[27] FutureID. WP34 - User Interface, 2013.

[28] FutureID. WP43 - Trust Services, 2013.

[29] FutureID. WP43 - Trust Services, D43.4 Implementation, 2014.

[30] FutureID-DOW. FutureID Grant agreement, Annex I - "Description of Work", 2012. URL https://dms-prext.fraunhofer.de/livelink/livelink.exe/overview/2625944.

[31] B. Haas. IAIK-PADES 1.18 SDK, 2013.

[32] C. Hanser. IAIK ECCelerate SDK 2.15, 2013.

[33] C. Hanser and D. Slamanig. Blank Digital Signatures. In *8th ACM SIGSAC Symposium on Information, Computer and Communications Security (AsiaCCS)*, pages 95–106. ACM, 2013. ext. Version: Cryptology ePrint Archive, Report 2013/130.

[34] M. Horsch, D. Derler, C. Rath, H.-M. Haase, and T. Wich. Open Source für europäische Signaturen — Vertrauenswürdige Basis für die elektronische Signatur. *Datenschutz und Datensicherheit*, 4, 2014.

[35] K. Lanz and H. Bratko. IAIK XAdES v1.3.2_1.16 SDK, 2008.

[36] K. Lanz and H. Bratko. IAIK-XSECT 1.15 SDK, 2008.

[37] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. Online Certificate Status Protocol - OCSP, Jun 2013. URL https://tools.ietf.org/html/rfc6960.

[38] OASIS Digital Signature Services TC. Digital Signature Services v1.0 — DSS Core Protocols, Elements, and Bindings v1.0, Apr 2007. URL http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.html.

[39] OASIS Digital Signature Services TC. Advanced Electronic Signature Profiles of the OASIS Digital Signature Service Version 1.0, Apr 2007. URL http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.html.

3 Project Description

The *FutureID* project seeks to build a comprehensive, flexible, privacy aware and ubiquitously usable identity management infrastructure for Europe, which integrates existing eID technology and trust infrastructures, emerging federated identity management services and modern credential technologies to provide a user-centric system for the trustworthy and accountable management of identity claims.

The *FutureID* infrastructure will provide great benefits to all stakeholders involved in the eID value chain. Users will benefit from the availability of a ubiquitously usable open source eID client that is capable of running on arbitrary desktop computers, tablets and modern smart phones. *FutureID* will allow applications and service providers to easily integrate their existing services with the *FutureID* infrastructure, providing them with the benefits from the strong authentication offered by eIDs without requiring them to make substantial investments.

This will enable service providers to offer this technology to users as an alternative to username/password based systems, providing them with a choice for a more trustworthy, usable and innovative technology. For existing and emerging trust service providers and card issuers *FutureID* will provide an integrative framework, which eases using their authentication and signature related products across Europe and beyond.

To demonstrate the applicability of the developed technologies and the feasibility of the overall approach *FutureID* will develop two pilot applications and is open for additional applications, which want to use the innovative *FutureID* technology.

FutureID is a three-year duration project funded by the European Commission Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318424.

4 Introduction

The eSign Services Add-on of the *FutureID* client is intended to provide means for a transparent signature creation using OASIS DSS [38], and, thus, abstracting the complexity of creating signatures in various formats and conforming to various standards [11, 12, 13, 15] and European regulations [9] from applications requiring to create signatures. Thereby, the Add-on should be flexible enough to support other signature formats and request formats, respectively.

In this document an in-depth description of the architecture of the eSign Services is given. Furthermore, the implementational details we consider important and the *FutureID*-specific additions to the profiles are discussed. The source code, developed along with this deliverable can be found in the `feature/eSignServices` branch of the *FutureID* repository.

4.1 Client

Since the eSign Services Add-on depends on the architecture of the *FutureID* client in general and on the architecture of the Add-on framework in particular, a description of the client architecture (adopted from [26] with slight modifications) is included for convenience of the reader.

As mentioned above, this section provides a brief description of the individual components and the architecture of the *FutureID* client. The components are illustrated in Figure 1. Please note that this deliverable only considers the eSign Services. For additional information about the other components please see the respective work packages.

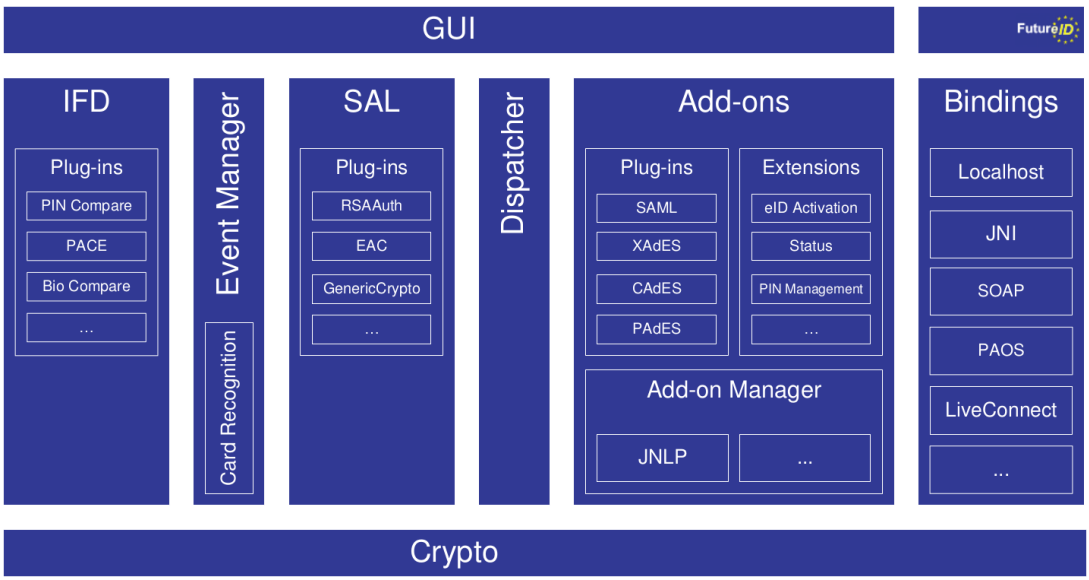

Figure 1: Architecture of the *FutureID* client

The Interface Device (IFD) is part of WP31 and specified in D31.2 [22]. The Event Manager, the Service Access Layer (SAL), and the Add-on framework belong to WP32. The signature

components, as illustrated as the CAdES, XAdES, and PAdES Add-Ons in Figure 1, are part of the eSign Service in WP33 and are 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.

4.1.1 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 do not need to 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 [22].

4.1.2 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 [23].

4.1.3 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 [23].

4.1.4 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.

4.1.5 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 [23].

4.1.6 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.

4.1.7 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.

4.1.8 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).

4.2 Outline

The remainder of this document is structured as follows. In Section 5 the architecture of the eSign services is introduced. Subsequently, in Section 6 we present our OASIS DSS profile, tailored to the requirements of *FutureID*. Then, in Section 7 we present our approach for viewing documents to be signed in a trusted manner and finally conclude with the presentation of a prototypical implementation of the graphical user interface in Section 8.

5 eSign Services Architecture

Owing to the fact that the basic architecture of the eSign Services was already introduced in [26], we use the section describing the architecture as a basis for this section and augment it with implementation details when necessary.

The eSign Services of the *FutureID* client consist of one or more `SignaturePlugin` implementations which comply with the Add-on framework specification in D32.3 [23], i.e., the `SignaturePlugin` implements the `ApplicationPluginAction` interface, which enables the communication with other components over bindings[1]. `SignaturePlugin` implementations forward the requests to implementations of the `SigningService` or `ValidationService` interfaces, respectively[2]. These exchangeable services, in turn, make use of `SignatureFormat` implementations for creating/verifying signatures in the various supported signature formats, like XAdES [15], CAdES [11], and so forth. Thereby, each `SignatureFormat` implementation constitutes a single component, enabling the support for additional signature formats.

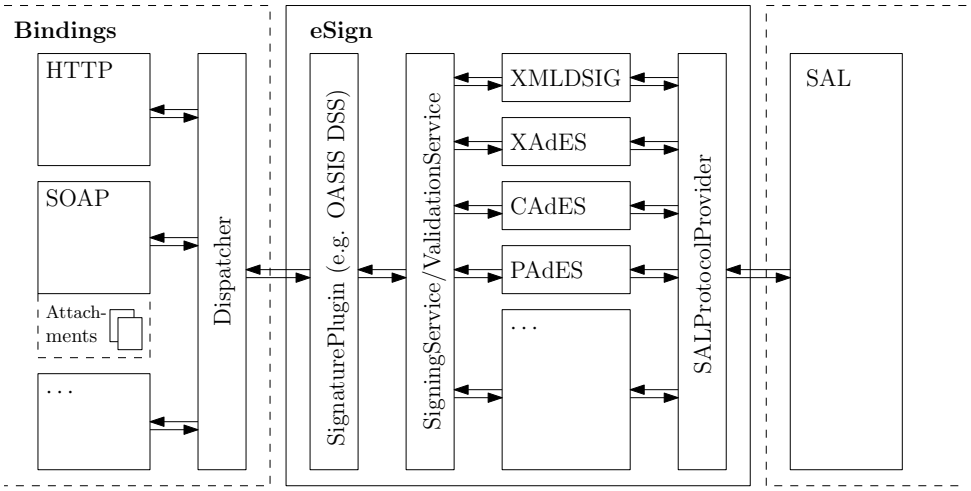

Figure 2: Componentized Overview of the eSign functionality of the *FutureID* client (adapted from [23])

Since the client is required to make 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, the SAL is made available over a Java Cryptographic Provider which provides the implementations making use of the required SAL protocols using the conventional Java Cryptography Architecture (JCA) provided factories, e.g., a `Signature` implementation wrapping a SAL protocol for signature creation on a smart card.

[1] The easiest way of interacting with the eSign Services is to send OASIS DSS requests within the body of a HTTP request to the localhost binding, i.e., `http://localhost:24727/eSign`.

[2] By means of a so called `SigningServiceProxy`/`ValidationServiceProxy` implementations.

5.1 Signature Plugin

The eSign Services of the *FutureID* client are required to support the OASIS DSS standard [38], while the design should allow for providing implementations of arbitrary other frameworks for signature creation and validation. Therefore, each SignaturePlugin uses abstract `Request-`, `Response-` and `ExceptionTransformers` for transforming the requests to an internal container format and vice versa. This way, one can easily support other request formats by simply providing alternative `Transformer` implementations. However, for the rest of this document we only consider an implementation of OASIS DSS together with various signature formats.

Regarding signature creation and verification, the Sign- and Verify-Requests are delegated to a `SigningServiceProxy` and a `ValidationServiceProxy`, respectively, which allows for using arbitrary `SigningService` and `ValidationService` implementations. By default we provide an implementation for local generation of XAdES, CAdES and PAdES BES signatures, while providing a web-service based validation service[3], which is also capable of extending the respective signatures to a format being suited for long term validation [14]. Since the signing and validation routines are encapsulated by the `SigningServiceProxy` and by the `ValidationServiceProxy`, respectively, our architecture allows for many different implementations while requiring minimal changes. As shown in Figure 3, one is able to provide signing services supporting arbitrary credentials, i.e., local signature generation and remote signature generation based on different setups. Further, we note that this architecture also enables the realization of so-called *on-demand signatures* where the key and the corresponding certification request is generated at signing time. A detailed discussion of all potential scenarios can be found in [34].

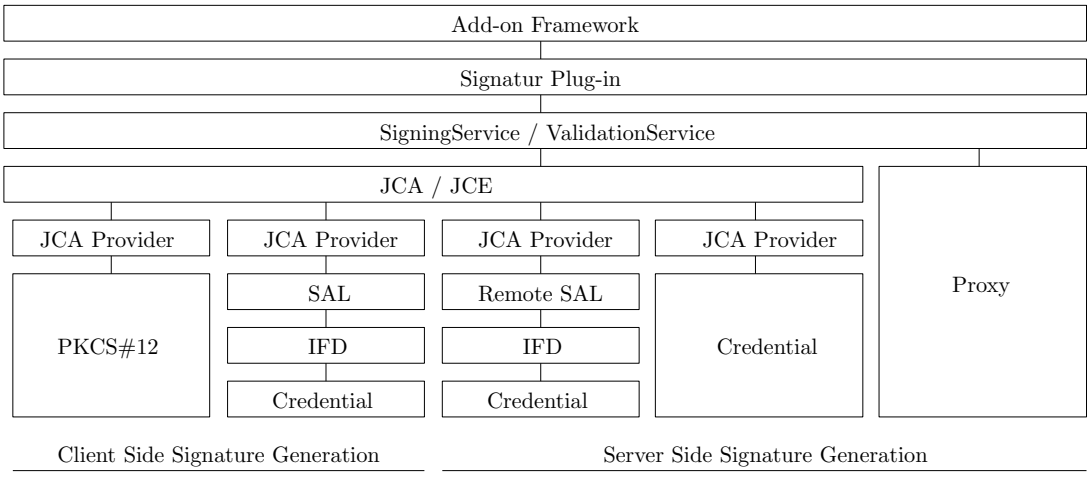

Figure 3: Possible implementations of Signature Plugins (adopted from [34])

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. While OASIS DSS does not define how to handle attachments in the former case, MIME at-

[3]We note that this validation services is developed in the TrustServices work package (WP43 [28]) and the WSDL can be found at http://trustservices.iaik.tugraz.at/trustServicesWeb/validation.wsdl

tachments, 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 signature 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, all signature services are implemented 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.

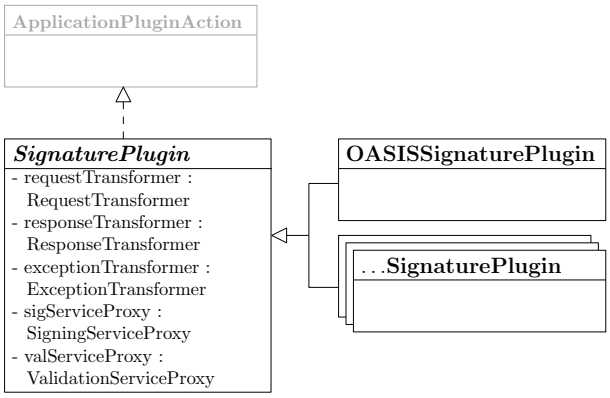

Figure 4: Signature Plugin

`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 architectures 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.

5.1.1 SAL Provider

As already mentioned before, a way for allowing libraries to use custom ways of accessing the signing and verification credentials, e.g., creating a signature on a smart card, is required. Our implementation enables 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 is a provider, providing implementations which interact with a SAL protocol by using the dispatcher. This provider allows for generating a `KeyPair` for SAL interaction by means of a `KeyPairGenerator`. Both, the `SALPrivateKey` and the `SALPublicKey`, respectively, can, thereby, be seen as references to the original private and public keys located in the credential. Furthermore, a so-called `ProviderInitProxy` is provided which abstracts the initialization of the provider and makes the public key and (a reference to) the private key available in an

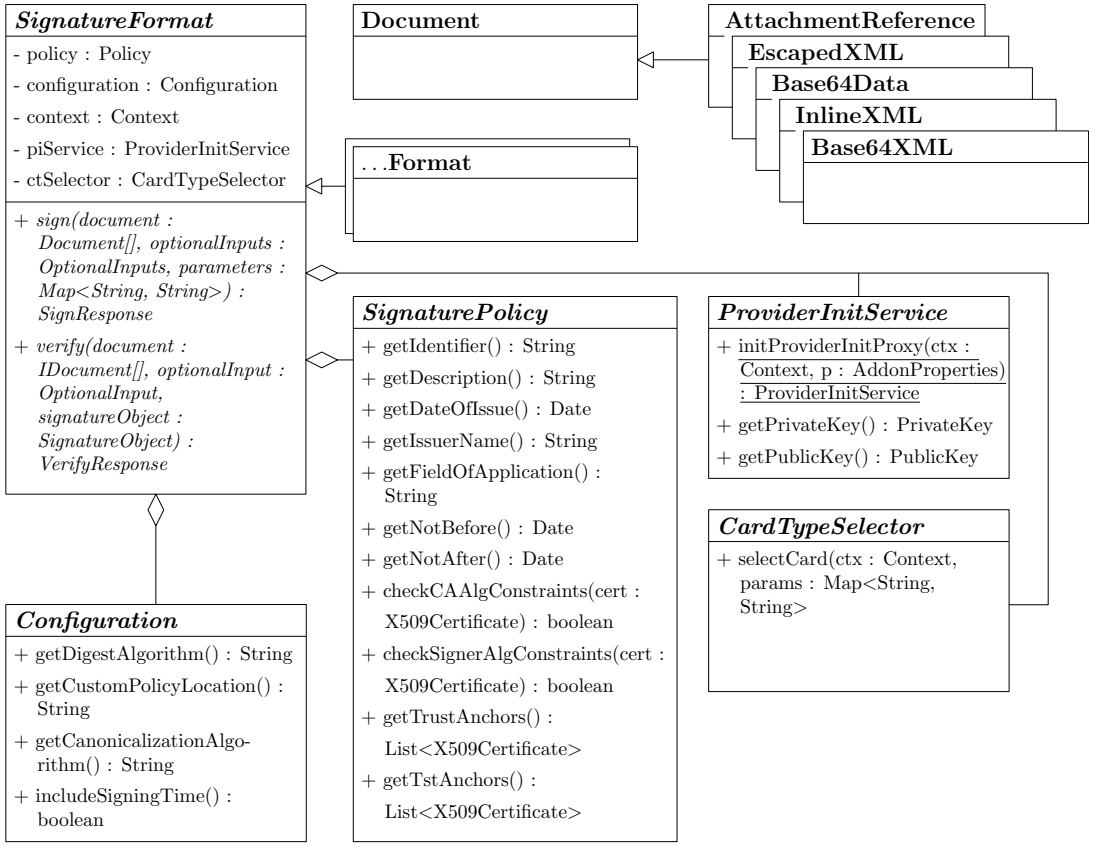

Figure 5: Signature Format Framework

implementation independent way. Since the SAL Provider is developed in D32.5, we do not further detail the implementation here, and assume a provider abstracting the signature creation using the SAL/IFD to be in place.

5.1.2 CardTypeSelector

In order to create signatures using various credentials, a strategy for selecting the desired credential among all available credentials is required. Our implementation encapsulates this strategy allowing for different ways of selecting credentials. The simplest case is to select the credential according to some supplied parameter, while one could implement arbitrary other selection strategies (see, e.g., Section 8.5). Furthermore, it is also required to provide the following selection strategies:

- `CardSelectionStrategy`: This selector allows for selection of the desired card when multiple cards of the selected card type (`CardTypeSelector`) are available. One simple strategy is to choose the first one, but, however, we also provide a prototypical GUI based strategy implementation (see Section 8.5).

- `DIDSelectionStrategy`: The selected card might provide multiple different DIDs corresponding to different keys. This strategy provides an abstract interface for selecting the DID. Again, a GUI based implementation of this strategy is used (see Section 8.5).

5.1.3 Configuration Parameters

The eSign Services are designed in a way, such that they make use of reasonable default parameters and can, thus, be deployed without any further configuration. However, experienced users or developers can also configure the eSign Services. This is done by making use of the Add-on properties file which is by default provided for any Add-on conforming to the Add-on framework [23]. That is, a properties file containing the configuration and being automatically loaded and passed to the eSign Services when the Add-on is loaded. This file needs to be located at `$HOME/.openecard/.addons/storage/eSign␣Services.properties`. The following table provides a list with possible configuration parameters and their description:

Parameter	Description
signingServicePath	The path to a jar file containing an implementation of an `org.openecard.esign.signing.SigningService`
signingServiceClass	The full qualified class name of the `SigningService` implementation
validationServicePath	The path to a jar file containing an implementation of an `org.openecard.esign.validation.ValidationService`
validationServiceClass	The full qualified class name of the `ValidationService` implementation
validationServiceUrl	The URL of the validation web service to be used.
providerInitServicePath	The path to a jar file containing an implementation of an `org.openecard.esign.signing.ProviderInitService`
providerInitServiceClass	The full qualified class name of the `ProviderInitService` implementation
viewerClassPath	The path to a jar file containing an implementation of an `org.openecard.esign.DocumentViewer`
viewerClass	The full qualified class name of the `DocumentViewer` implementation

Table 1: Configuration Parameters of the eSign Services

As we can see, one can provide own implementations for SigningServices, ValidationServices, ProviderInitializationServices and DocumentViewers if the default implementations do not fulfill the requirements of a certain use-case. As mentioned before, when one of the parameters is not present in the properties file, the eSign Services use the default implementations.

5.1.4 Required Changes w.r.t. D33.2 [26]

- There is an other intermediate layer between the `SignaturePlugin` and the various `SignatureFormat` implementations, i.e., `Signing/ValidationService`. This ensures more flexibility regarding future developments.
- The SALKey is not required to keep a reference to the respective SAL protocol since the interaction with the SAL and arbitrary protocols can be performed using the dispatcher, which is part of the Add-on `Context`.
- We do not require a singleton `SignatureFormatRegistry`, since the selection of the signature format is done using the `dss:SignatureType` in the OASIS DSS requests and responses. This leads to a more stateless, and, thus, more flexible solution.
- The `sign()`-method of the `SignatureFormat` does no longer require an OutputStream to write the resulting signature to, since the signature is passed within the `SignResponse`.

5.2 Potential Risks and Countermeasures

Basically, we assume that an attacker has no direct access to a user's machine, and, thus, we do not consider changes in the configuration files in order to, e.g, specify an alternative SigningService, as an attack. However, since the eSignServices are deployed as an Add-on to the *FutureID* client it is important to ensure the integrity of this Add-on when downloading them from some external source, which can for instance be achieved by signing the Add-on jar file or by authentically providing hash values, which can be used for checking the integrity of the downloaded files.

Assuming the integrity of the eSign component, the documents to be signed will always be displayed to the user in a trusted manner (cf. Section 7) before signing (unless the user explicitly switches this feature off). Furthermore, for each document to be signed, a confirmation of the user is required. Consequently, no signature will be issued without the user's interaction.

6 OASIS DSS

As mentioned before, the *FutureID* client responds to OASIS DSS [38] requests, received over the bindings framework. In this section we firstly outline our design decisions regarding our OASIS DSS implementation and then introduce the *FutureID* specific profile. Basically, our design decisions resemble the requirements specified in D33.1 - Requirements Report [25]. For example OASIS DSS requests, illustrating the various scenarios described subsequently, we refer the reader to the `demo` folder in the `eSignServices` project, i.e., we do not explicitly provide examples in this document.

Firstly, we note that according to OASIS DSS, input documents in `dss:InlineXML` format are exclusively canonicalized before signing in order to gain maximum robustness against subsequent changes of documents due to marshalling and unmarshalling. However, an exclusive canonicalization changes the document, while preserving semantic equality. We, thus, note that when such a document is returned with the `dss:SignResponse`, it might be slightly modified, i.e., at least the unneeded namespaces defined in the document to be signed are cut off. However, since exclusive canonicalization is used, this does not influence the validity of the signature. Nevertheless, we recommend using `dss:EscapedXML` or `dss:Base64XML` instead, since these types are more robust against the flaws described above.

Moreover, for similar reasons, we always return the signatures as `dss:Base64Signature` contained in a `dss:SignatureObject` in case of detached and enveloping XAdES signatures. In case of enveloped XAdES Signatures, we return a document containing `dss:Base64XML` which is referenced in the `dss:SignatureObject`. In case of PAdES and CAdES a `dss:Base64Data` document, referenced in the `dss:SignatureObject`, is returned.

The following itemization provides an overview of the currently supported elements. Support for other elements is currently not required, while, due to our architecture, they can easily be supported by adding transforming routines to the respective transformers.

- `dss:SignRequest`
  - `RequestID` attribute
- `dss:VerifyRequest`
  - `RequestID` attribute
- `dss:SignatureObject`
  - `dss:Base64Signature`
  - `dss:SignaturePointer`
- Documents
  - `dss:InlineXML`
  - `dss:Base64XML`
  - `dss:EscapedXML`

- `dss:Base64Data`
- `dss:AttachmentReference`

- OptionalInputs
  - `dss:SignatureType`

    XAdES: `urn:ietf:rfc:3275`

    CAdES: `urn:ietf:rfc:3369`

    PAdES: `urn:futureid:esign:pades`

    BDS: `urn:futureid:esign:bdss` (cf. Section 6.2.5)
  - `dss:IncludeObject`
  - `dss:SignaturePlacement`
  - `dss:SignedReferences`

Furthermore all elements defined in the following section are supported as well.

6.1 *FutureID* Profile

In this section we introduce the *FutureID* profile of the OASIS DSS core schema [38] and discuss the request format being supported by the eSign Services. The *FutureID* profile is shown in Listing 1 and discussed subsequently. All additional elements are intended to be supplied as `dss:OptionalInput`. Although, our current implementation does not require elements from the AdES Profile [39] of OASIS DSS, our marshalling and unmarshalling routines support this elements in case they are needed in the future.

Listing 1: *FutureID* OASIS DSS Profile

```
<xs:schema xmlns="urn:eu:futureid" xmlns:ades="urn:oasis:names:tc:dss:1.0
    :profiles:AdES:schema#" xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xs="http://www.w3.org
    /2001/XMLSchema" xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
    xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" xmlns:vr="
    urn:oasis:names:tc:dss:1.0:profiles:verificationreport:schema#"
    targetNamespace="urn:eu:futureid" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xs:import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="
      xmldsig-core-schema.xsd"/>
  <xs:import namespace="urn:oasis:names:tc:SAML:1.0:assertion" schemaLocation="
      oasis-sstc-saml-schema-protocol-1.1.xsd"/>
  <xs:import namespace="http://uri.etsi.org/01903/v1.3.2#" schemaLocation="XAdES.
      xsd"/>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="xml
      .xsd"/>
  <xs:import namespace="urn:oasis:names:tc:dss:1.0:core:schema" schemaLocation="
      oasis-dss-core-schema-v1.0-os.xsd"/>
  <xs:import namespace="urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#"
      schemaLocation="oasis-dss-profiles-AdES-schema-v1.0-os.xsd"/>
```

```
  <xs:element name="counterSignature" type="xs:integer"/>
  <xs:element name="policy" type="dss:InlineXMLType"/>
  <xs:element name="extendTo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="extendToFormat" type="xs:anyURI"/>
        <xs:element name="tspServer" type="xs:anyURI"/>
        <xs:element name="tspPort" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="validationType" type="xs:anyURI"/>
  <xs:element name="wrapInASiCContainer" type="xs:boolean"/>
</xs:schema>
```

6.1.1 counterSignature

Although a counter signature can simply be created by means of OASIS DSS, we provide an alternative way for directly creating counter signatures on a document. This tag is intended to be set to the number of required counter signatures n and is intended to be omitted when no counter signature is required. Upon signature creation the user is asked to provide n distinct credentials. This element is allowed for `dss:SignRequest`s.

6.1.2 policy

This element is intended to contain a policy conforming to Section 6.3 which should be used for signature creation/verification.

6.1.3 extendTo

The `extendTo` field can turn up in the optional inputs of a `dss:VerifyRequest` and allows for requesting an extension of a signature to a long term format. Thereby the `extendToFormat` can be set to one of the following URNs:

- `urn:futureid:ades:lt`
- `urn:futureid:ades:lta`

Furthermore, `tspServer` and `tspPort` allow for specifying a timestamping server URI and port.

6.1.4 validationType

Since long term validation according to [14] is supported for XAdES signatures, one can make use of this element in `VerifyRequests`. The allowed URNs are

- `uri:etsi:org:102853:v1.0:ValidationType:BASIC`
- `uri:etsi:org:102853:v1.0:ValidationType:ADEST`
- `uri:etsi:org:102853:v1.0:ValidationType:LONGTERM`

In case AdES-T or long term validation is not supported for a signature format, the basic validation is performed, independently of the requested validation type.

6.1.5 `wrapInAsicContainer`

This optional input for `dss:SignRequests` controls whether the response is wrapped within an ASiC container [16].

6.2 Signature Formats

The signature formats required by the eSign Services are created using the IAIK libraries for XML, CMS, and PDF signatures [18, 31]. In the following sections we give an overview of what is currently supported and what is planned for the future.

6.2.1 XAdES

The IAIK XAdES library [35] is capable of generating full-fledged XAdES signatures. We use it for locally generating XAdES BES signatures according to the XAdES baseline profile [2] and remotely validating and creating baseline profile conformant XAdES LT and LTA signatures (Trust Services [28]).

Thereby the OASIS DSS elements are used as specified in the standard. Upon validation, the status together with a detailed message, making the validation process more comprehensible for the user, is provided (cf. [29])

6.2.2 PAdES

Currently the IAIK PAdES library is capable of creating and validating PAdES BES signatures according to [3]. Since the PAdES format is not covered by OASIS DSS core, we require that exactly one PDF document as `dss:Base64Data` or as `dss:AttachmentReference` with corresponding SOAP Attachment is supplied. Since the return format was already discussed above, it is not separately discussed here.

However, library support for LT and LTA formats is expected within the next few months and will then be integrated in the eSign Services.

6.2.3 CAdES

Similarly, also the IAIK CAdES library is currently capable of creating and validating CAdES BES signatures according to [4]. Here, one is required to supply one document of arbitrary OASIS DSS document type. However, since CMS signatures are already discussed in [38] we do not further discuss them here.

As above, library support for LT and LTA formats is expected within the next few months.

6.2.4 Integration of External Tools for Signature Creation

In order to illustrate the flexibility of our implementation, we implemented two alternative `SigningServices`. The first `SigningService` seamlessly integrates the Austrian Mobile Signature (supporting XML and CMS Signatures) as discussed in [21], while the second wraps the PDF over[4] tool, an existing tool written in Java and allowing to sign PDFs using the Austrian citizen card or the Austrian mobile signature, respectively.

By simply setting the `signingServiceClass` property in the Add-on properties to `org.openecard.esign.at.mobile.MobileSigningService` or `org.openecard.esign.at.mobile.MobilePdfOverSigningService` this alternative signing services are used instead of the default signing service.

6.2.5 Integration of Non-Conventional Digital Signatures

Quite recently, the concept of Blank Digital Signatures (BDS) [33] was introduced. In a nutshell, BDS allow an originator to delegate the signing rights for a so called *template* (containing fixed and exchangeable elements) to a proxy. The designated proxy is then able to choose one element for each of the exchangeable elements and issue a signature on behalf of the originator on this so called *instance* of the template. Once the proxy signed such an instance, everyone is able to validate the signature, while not learning anything about the unused choices in the exchangeable elements. In [20], a Java based implementation of BDS was presented together with some optimizations of the BDS scheme. In this work, also an XML based container format for templates and messages was proposed.

Based on the observation that the template signature in BDS is modeled by a conventional digital signature on the description of the template, together with an identifier of the proxy's public key (certificate) and also the instance signature is computed in a similar way, we tried to integrate BDS as a `SignatureFormat`. Surprisingly, no changes to the implementation and only some minor changes to the container format were required in order to do so (the container format additionally required an identifier for the used signing algorithm of originator and proxy).

From an OASIS DSS point of view we require the following request format. Upon creating the template signature exactly one document with the attribute `ID` set to `"template"` is required. The document can, thereby, be supplied as `dss:Base64XML`, `dss:InlineXML` or

[4] http://webstart.buergerkarte.at/PDF-Over/

`dss:EscapedXML`, respectively. Upon template signature generation the `SignatureFormat` implementation then firstly requests the credential of the proxy in order to obtain its public key certificate. Then, one needs to supply the originator's credential in order to create the template signature, which is then returned as `Base64XML` contained in a `dss:DocumentWithSignature` and a `dss:SignaturePointer` pointing to it.

Upon generation of the instance signature, one needs to supply the signed template and a second document representing the instantiation, i.e., a document of type `dss:Base64XML`, `dss:InlineXML` or `dss:EscapedXML` with the `ID` attribute set to `"message"` and compatible with the provided template. The signature format then requests the proxy's credential and issues an instance signature if the instantiation is compatible with the supplied template.

6.3 Signature Policies

Signature Policies can be used to specify certain constraints regarding the creation and validation of signatures. Currently we support the following constraints:

- Constraints defining the allowed algorithms,
- constraints defining the minimum key length for certain algorithms, and
- constraints regarding the trusted certificates for signatures and timestamps, respectively.

As input format for policies, the format standardized in [1] is used. Thereby, the constraints regarding the allowed algorithms need to be specified as `SignerAlgConstraints` elements, being child elements of the `AlgorithmConstraintSet` element. Furthermore, the trustanchors can be supplied using the `SigningCertTrustConditions` and the `TimeStampTrustConditions` elements. An example policy can be found in Appendix A.

If no policy is supplied in the request, the policy of the server is used. That is, a policy defining the minimum key lengths and allowed algorithms and containing the trustanchors for signatures and timestamps according to the trusted certificates of the Trust Services [28].

6.3.1 Changes to the Schema defined in [1]

Currently, we do not require all elements of the policy format defined in [1], and, thus some changes to the schema were made, i.e., some required elements were set to optional. We outline those elements subsequently:

- Firstly, we note that we can use XML signatures, which are provided by our framework, if authenticity and integrity of a policy is required. We, thus, do not require policy elements to be explicitly digested, and, thus the following elements are set to optional:
  - `SignPolicyDigestAlg`
  - `SignPolicyDigest`

- Currently, commitment types are not supported. Thus, we set the `CommitmentRules` to optional.
- Furthermore, both, certificate revocation lists and OCSP responses, are considered by the validation web service [28]. Thus, we do not need the elements defining the requirements for the revocation checks, i.e., `crlcheck`, `ocspcheck`, ..., and, consequently, set the `SignerRevReq` element to optional.

6.4 ASiC Signature Containers

ASiC (Associated Signature Container) is in general a kind of zip-archive (container) which summarizes signed documents and the appropriate/used signature information in a predefined manner [16]. We provide means for wrapping signed documents into ASiC containers, as well as means for unwrapping documents and signatures from ASiC containers.

6.4.1 SignRequests with ASiC

As already discussed above, the `SignRequest` can have an optional input parameter (`OptionalInputs`), which defines if the result of the signing process should be delivered as ASiC container or not. If an ASiC-Response is requested, the *FutureID* client needs to wrap the `SignResponse` into ASiC-Format. For this purpose, the signed documents are transferred in the container. The matching/corresponding signatures are stored in a predefined signature file (META-INF/signatures.xml). By adhering to this archive structure it is ensured that other applications implementing ASiC, can further process so-created archives.

6.4.2 VerifyRequest with ASiC

Also the `VerifyRequest` allows to hand over an ASiC container to verify its content via the `SignaturePlugin`. To meet this requirement, it is necessary to unwrap the given ASiC container into its components and create a corresponding `VerifyRequest`. This is done by unpacking the container elements and transforming them into the needed input parameters for further processing.

7 Trusted Viewer

Basically, a trusted viewer is intended to provide a secure and trustful view on digital documents. Thus, it is an integral requirement to realize a reliable document signing function within the *FutureID* client. Thereby, the viewer has to make sure, that the signer of a digital document is able to see the whole document content, which, in turn, enables the signer to make a reasonable decision whether he wants to issue a signature on the viewed document(s) or not. The signer should be able to read and inspect the document content carefully (including the small prints etc.) and sign it afterwards. As mentioned before, the biggest challenge for a trusted viewer is the identification and visualisation of the entire content within the document (regardless of its metadata and formatting). If there occurs for example invisible text (maybe white text on a white background) within a text document and the user signs this document, he has also signed non-readable part of the document, while possibly only being aware of the readable part. Thus, the main goal of a trusted viewer it to prevent signing of unknown and uncontrollable content (for the user/signer).

7.1 Implementation Details

Since the *FutureID* client should not be bound to a particular platform, we decided to provide a mechanism for easily exchanging the trusted viewer implementations by means of dynamically loaded jar files. To support this, a so-called `DocumentViewerProxy` is integrated in the `SignaturePlugin`. This proxy then loads the viewer implementation, specified in the `AddonProperties`.

In the following chapters, we introduce our trusted viewer concept, building on the Java Swing library and using a simple JFrame to display the required content. With this approach it is possible to develop a viewer which is compatible with Windows, Linux and Mac systems and provides a similar look and feel for the user. We, however, note that we currently do not provide a trusted viewer for mobile platforms.

In order to represent the document content in a human-readable form, one may additionally supply a kind of template, i.e., a stylesheet. This stylesheet defines the structured representation of the document data so that the reader can simply grasp the content of the document. In case of PDF documents, these representation structure is defined by the document itself. In contrast, an XML document does not contain such style information by default, and, thus, may need an attached/integrated style sheet to transform its data in a human-readable representation. To ensure this transformation and make it permanently reliable, comprehensible and repeatable, it is essential to also sign this style sheet. Thus, our implementation assumes the style sheet to be provided within a `ds:Transform` element of a `dss:SignedReference` element of the `dss:SignRequest`, which is then also contained in the respective `ds:Reference` of the signature. An example `dss:SignRequest` containing style information is provided in Appendix B.

From an implementation point of view, the trusted viewer implements the `DocumentViewer` interface, so after calling the defined `view()`-method, the necessary style and content information

is parsed from the given input. In dependence of the submitted file type, we need to have different implementations to parse, proceed and afterwards render the document correctly.

7.2 Trusted XML Viewer

Like mentioned before, here it is possibly to additionally supply an XSL style sheet as an OASIS DSS `InputDocument`, which is then also signed together with the other supplied documents. With these style definitions, we are able to transform the document via XSLT into a complete/entire and easy to use HTML representation which can be rendered in the trusted viewer with Swing functionality. If no style sheet is embedded, the viewer is not able to perform this transformation to a formatted graphical representation. In this case we also use Swing functionality to render the XMLs content as plain text, thus the user can inspect his document anyway.

7.3 Trusted PDF Viewer

A reproducible and reliable presentation of any PDF file for signing purposes is not achievable, because of several PDF functions which allow or enable to integrate dynamic content into the document, e.g., embedded JavaScript or the use of external references to additional content. The possible usage of transparent objects in the document also prevents, that the user is able to see and control the whole document information before signing it. To circumvent these problems Adobe developed the PDF/A standard [6][5], which makes several restrictions regarding the used PDF functions. For these PDF/A documents every PDF/A conformant viewer is a trusted viewer. Though, the precondition that any PDF file must be PDF/A compliant for signing purposes is not worthwhile. So our approach is to analyze the given file to identify content (PDF elements), which prevents a reliable representation, e.g., a document reloading dynamic content from external references. To do so, we decided on using Apache PDFBox to realize analysis and rendering of the given document(s). This decision is based on the fact, that PDFBox is distributed under APL V2.0 (which meets the target of an OSI-approved open source license) and it provides a good functional range to the implement the requirements. We are able to use the built-in PDF/A-Validation to inspect the given file. Afterwards we filter the result-list of this validation to identify elements which prevent a reliable and trustful representation of the document content.

Table 2 lists our restrictions for a reliable presentation of PDF documents (it is a subset of PDF/A-1b restrictions).

To ensure a platform independent representation of the given document(s), we also use PDFbox functionality in conjunction with the Swing library. This reduces the project dependencies and simplifies implementation and maintenance. Within an extra GUI area we show additional information of the document (e.g. meta data like filename, lastmodified date, filesize) as well as the results of our content analysis. If the processed file contains an element from our restriction

[5] We did not use the newer PDF/A-2 [7] or PDF/A-3 [8] Standards, because they are less restrictive in dealing with layers, transparencies and embedded objects, which is no optimal evolution for signing purposes. A PDF/A compliant document also satisfies the PDF/A-2 and PDF/A-3 restrictions.

Restriction	Substantiation
Whole font information must be embedded	Ensures that all contained signs and characters can be displayed properly.
No embedding of external content via references (pictures, fonts, pages)	It is not possible to ensure that external content is static (unchanged) and long-time available.
No usage of alternative pictures	It is not possible to ensure which alternative picture was displayed during signing process.
No usage of JavaScript	JavaScript enables to integrate dynamic content into the file (e.g., interaction with web services to load additional information, load data from databases, use multimedia functionality). It is not possible to ensure unchanged/static content over time.
No multimedia objects embedded or referenced (3D-Objects, Video, Audio)	Referenced objects are not long-time static and available. Viewer partially use external players to display/play multimedia objects. Availability and reliable implementation of external players is not guaranteed.
Non-existence of document layers	Possibly the signer is unaware of the existence of another document layer with potentially other or further information. So he is not able to inspect this content properly. Furthermore it is not possible to ensure which document layer was displayed during signing process.

Table 2: Restrictions for Reliable PDF Document Presentation

list we display a suitable warning, alerting the user that it is not recommended to sign this file. This note also includes an explanation / hint which dynamic elements were found and why they should not be signed.

8 Graphical User Interface

For demonstration purposes, a prototypical graphical user interface was implemented, which is intended to be further developed and aligned with the *FutureID* GUI strategy in the GUI workpackage [27]. In this section we want to give an overview of this GUI implementation. While most of the contents of this section are kept quite high-level, in some cases implementation details are shown to make this document more comprehensible and to demonstrate how the existing *FutureID* architecture and infrastructure was leveraged for these tasks.

The eSign GUI requirements can be grouped into several aspects:

1. Graphical confirmation and information dialogs often needed (after a sign- or verification request was sent and processed by the eSign Add-on) to request additional information from the user, e.g., to choose an appropriate card or a DID (see Section 8.4)
2. Provision of an (easy to use) interface to allow the user to create sign- and verification requests and transmit them to the eSign Services server (cf. Section 8.3).
3. Creation of Oasis DSS conformant sign requests, transmitting them to the server, parsing the response and saving the resulting signature.
4. Creation of Oasis DSS conformant verification requests, transmitting them to the server, parsing the response and showing the verification results.

8.1 GUI Architecture for Dialogs Requesting User Input

The GUI functionality of the *FutureID* architecture is enclosed in an independent layer (see Figure 1), which allows for the support of multiple client platforms, while requiring only little platform specific code. In particular, we use the `obtainUserconsent`-method of the `Context` object being available for each Add-on. This method allows for obtaining a `UserConsent` object, which is the central class for creating and managing standardized user dialogs with a consistent look and feel. The content of these dialogs can, in turn, be divided into multiple `Steps`, which can be used for navigation. If the user navigates to the next `Step`, a `StepResult` is returned. This `StepResult` contains the GUI and the user input, which can then be used for further processing.

8.2 Loading Configuration Parameters via the Add-on Framework

The Add-on framework of the *FutureID* client allows to dynamically load additional protocols, bindinds as well as certain parts of the GUI in a plug-in like concept (cf. [24]). The specific classes to be loaded as well as GUI definitions are defined in an XML structure in the Add-on's class-path. For GUI-related Add-ons the specification allows to use the so called `ConfigDescription` element to define and specify entries to be displayed in the *FutureID* client's configuration screen (cf. Figure 6, Figure 7).

```
<ConfigDescription>
    <Entries>
     <EnumEntry>
          <Key>esign.format</Key>
          <Value>XAdES</Value>
          <Value>CAdES</Value>
          <Value>PAdES</Value>
          <LocalizedName xml:lang="DE">eSign-Format</LocalizedName>
          <LocalizedName xml:lang="EN">eSign format</LocalizedName>
          <LocalizedDescription xml:lang="DE">Auswahl des per default genuzten eSign-Formats</LocalizedDescription>
          <LocalizedDescription xml:lang="EN">The default eSign format to be used</LocalizedDescription>
    </EnumEntry>

    <ScalarEntry>
        <Key>TimestampServer</Key>
        <Type>String</Type>
        <LocalizedName xml:lang="DE">Zeitstempel Server</LocalizedName>
        <LocalizedName xml:lang="EN">Time Stamp Server</LocalizedName>
        <LocalizedDescription xml:lang="DE">Zeitstempel Server</LocalizedDescription>
        <LocalizedDescription xml:lang="EN">Timestamp Server</LocalizedDescription>
    </ScalarEntry>
     <ScalarEntry>
        <Key>TimestampServer.Port</Key>
        <Type>String</Type>
        <LocalizedName xml:lang="DE">Zeitstempel Server Port</LocalizedName>
        <LocalizedName xml:lang="EN">Timestamp Port</LocalizedName>
        <LocalizedDescription xml:lang="DE">Zeitstempel Port</LocalizedDescription>
        <LocalizedDescription xml:lang="EN">Timestamp Port</LocalizedDescription>
    </ScalarEntry>
   </Entries>
</ConfigDescription>
```

Figure 6: The XML Definition of the Configuration GUI

8.3 Signing and Validation

In order to also be able to demonstrate the singing and verification functionality by means of a GUI, a prototypical dialog was created and integrated into the existing RichClient implementation (cf. Figure 8). It was designed to be easily accessible (with only two mouse clicks), since we predict this to be a frequently used functionality. Thereby, the user can choose between signature creation and signature validation.

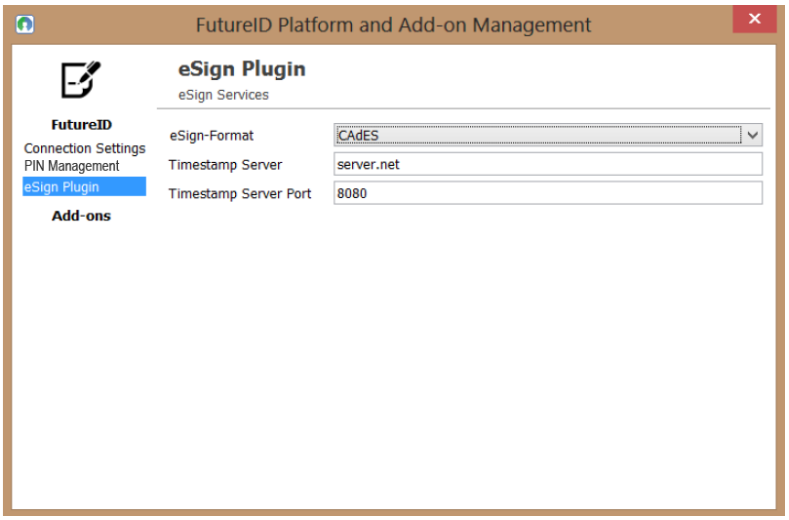

Figure 7: The Resulting Configuration Dialog for the eSign Services

8.3.1 Signing

The dialog for the signing functionality is shown in Figure 8. The user can choose the signature format to be used (XAdES, CAdES or PAdES), whether to wrap the result in an ASiC signature container and whether the returned signature will be enveloped by the document to be signed in case of XAdES and CAdES.

To select the file to be signed, a file selection dialog can be opened. If PAdES format is used, only Adobe PDF files are shown. Once a file was selected, the user can submit the sign request. If the sign request was unsuccessful, an error dialog with details of the error is shown. In case of an successful request, the response is parsed and the signature or the file containing the signature is extracted and can be saved to disk using a file save dialog.

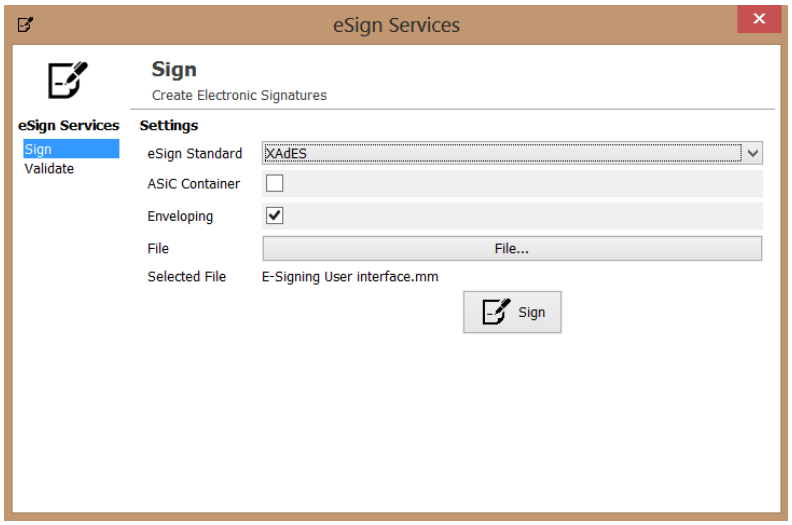

Figure 8: The Dialog for Creating Electronic Signatures

8.3.2 Validation

The appearance of the signature validation dialog is similar to the signing dialog, but has a different set of options for the user to select from. Again, a file chooser is employed to allow the user to select the signature file to be used for the validation request (see Figure 9). If the "Validate" button is pressed, a validation request is created and transmitted. The validation result is then shown to the user in an information dialog.

8.4 User Prompt Dialogs

When a signature request is recieved by the eSign Add-on, three things have to be decided:

1. Which card type is used for signing

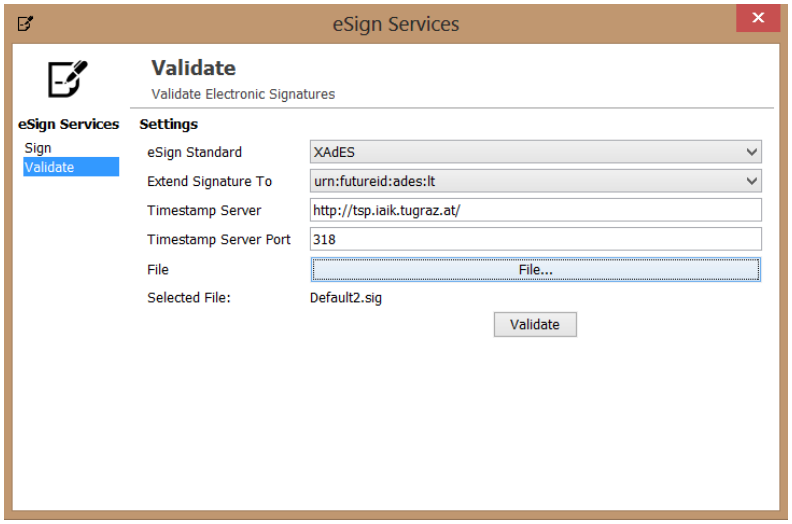

Figure 9: The Signature Validation Dialog

2. Which card is used for signing
3. Which DID is used for signing

These decisions are encapsulated within so called `Strategies` (see also Section 5.1.2), allowing for an easy exchange of the respective implementations. Technically, these are interfaces that can be implemented to fit different applications. For two of these strategies, we provide a GUI implementation, in which the user can make these decisions. Due to the easy exchangeability of these strategies, the eSign Services can later easily be aligned with the overall *FutureID* GUI [27].

The strategies are executed during run-time and in the following order:

`CardTypeSelectionStrategy`
: This strategy is used to specify and demand a certain type of card (e.g., German identity card) to be used for the signing process. Our implementation uses the card type information specified in the `OptionalInputs` structure of the sign request itself and does not provide a dedicated GUI. If no card type is specified in the `OptionalInputs` our `CardSelectionStrategy` will accept any card with signing functionality.

`CardSelectionStrategy`
: The `CardSelectionStrategy` is used to specify the card to be used for the signing process. Our implementation is as follows: If no card is available, we present a GUI to the user, requesting him to either supply (1) a card of the previously selected card type, or (2) any card that provides signing functionality if no card type was specified. In case a card, that matches the requirements is already available, we proceed using that card directly without any user prompt. If more than one card matching the requirements is available, again a GUI with a list of these cards is presented to the user.

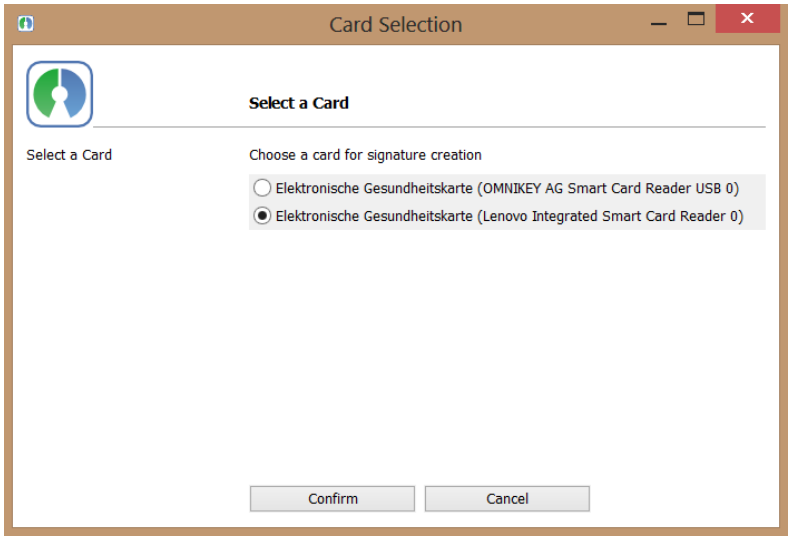

Figure 10: Dialog for Selection of a Card

`DIDSelectionStrategy`
: This strategy implements a GUI dialog to let the user select a DID from a list of DIDs stored on the previously selected card, in case that there are multiple DIDs available.

8.5 Localization

An extendable language and localization system is used in *FutureID* so that support for more languages can be achieved without much effort, since all text that is used in the application is defined in external language files in clear text (see for example Figure 11). Currently all GUI dialogs created for the eSign services are available in English and German language.

```
# English
addon.esign.dialog.choosecard = "Choose a card for signature creation:"
addon.esign.dialog.choosecard.contentdescription = Card Selection
addon.esign.dialog.choosedid.label = Select a DID
addon.esign.dialog.choosedid.stepname = DID Selection
```

Figure 11: Example of a Language Definition File for an eSign Dialog

A Example XML Policy

```
<SignaturePolicy xmlns="http://uri.etsi.org/2038/v1.1.1#" xmlns:xades="http://uri
    .etsi.org/01903/v1.3.2#">
    <SignPolicyInfo>
      <SignPolicyIdentifier>
        <xades:Identifier>http://www.futureid.eu/v.1.0</xades:Identifier>
        <xades:Description>test</xades:Description>
      </SignPolicyIdentifier>
      <DateOfIssue>2014-02-06T15:09:00</DateOfIssue>
      <PolicyIssuerName>IAIK, Graz University of Technology</PolicyIssuerName>
      <FieldOfApplication>FutureID eSignServices</FieldOfApplication>
      <SignatureValidationPolicy>
        <SigningPeriod>
          <NotBefore>2014-02-06T15:09:00</NotBefore>
        </SigningPeriod>
        <CommonRules>
          <SigningCertTrustCondition>
            <SignerTrustTrees>
              <CertificateTrustPoint>
                <TrustPoint>
                  <X509Certificate>
MIIEpjCCA46gAwIBAgIDAYBfMA0GCSqGSIb3DQEBCwUAMEQxCzAJBgNVBAYTAkRF
MRAwDgYDVQQKDAdnZW1hdGlrMSMwIQYDVQQDDBpnZW1hdGlrIExhYm9ydGVzdCBl
R0sgQ0EwMjAeFw0xMTA3MDgwMTIyMjVaFw0xMzA3MDgwMTIyMjVaMIGcMQswCQYD
VQQGEwJERTEgMB4GA1UECgwXZ2VtYXRpayBNdXN0ZXJrYXNzZTFHS1YxEzARBgNV
BAsMClgxMTAxMDMzOTMxEjAQBgNVBAsMCTk5OTU2Nzg5MDEbMBkGA1UEAwwSTGV0
aXppYSBOaWdnZW1leWVyMRMwEQYDVQQEDApOaWdnZW1leWVyMRAwDgYDVQQqDAdM
ZXRpemlhMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqC+s9boZkZcB
3Rq/aqRrXO1RorSpZU+K2ojxn19b3JZJ//RyWweow/5d4NG/kDYyDzbzyZ8pYWNl
+MdQ2Ye4jPvaw12KhUXeQmbW2ZPUeFLrIb9YZ4hL4ABC8w/WWaonrtO1LltjFvYf
UTKR2GacLsuMPidZFx2YLJxbOQ1GQBt5PC/f6d7pqXH3QX22mu4KRBLq82QOZnPY
2Pkcvih1XXzdnwQfSDrgXgxsm02Dn3ZU6/ckRqQcQYXq/L6ZqN1fRUjtBjHQFfXH
DmcS90fF3QXrDjyxKYCtmPvGrvLWX7cfcciPTLMU8nwhNhrGdpH0LWZw679x7PxJ
WO+VZuFAXwIDAQABo4IBRjCCAUIwEwYDVR0lBAwwCgYIKwYBBQUHAwIwEQYDVR0O
BAoECEDLvelPo06yMEQGA1UdIAQ9MDswCQYHKoIUAEwERjAuBgcqghQATARwMCMw
IQYIKwYBBQUHAgIwFRoTb2lkX3BvbGljeV9tdXN0ZV9jcDAwBgUrJAgDAwQnMCUw
IzAhMB8wHTAQDA5WZXJzaWNoZXJ0ZS8tcjAJBgcqghQATAQxMBMGA1UdIwQMMAqA
CE2ldD5PqylIMEMGCCsGAQUFBwEBBDcwNTAzBggrBgEFBQcwAYYnaHR0cDovL29j
c3AuZ2VtYXRpay5kZTo4MDgwL0NNT0NTUC9PQ1NQMA4GA1UdDwEB/wQEAwIHgDA2
BgNVHR8ELzAtMCugKaAnhiVsZGFwOi8vY3JsMDAuZ2VtYXRpay5pbnZhbGlkL25v
dC11c2VkMA0GCSqGSIb3DQEBCwUAA4IBAQCO/FykcCLLU3Q0lWYrOxVROwTTS3sg
M2YDH9/CnDQA1MNRAkJBeE2pwR1Ol/nlZQZJzHPtzuKvL18aj7NMIdqWYJ8Ijx+B
nfoUTNONONksxas0ySnwz2ztLv/3I0/A2VZw2wMXpeNHRj99taI1mLEjaHqdWcSA
L9etm7VCHW3TX4QekLPJfaTAfJhq0koMfouBWjVzFLZTGKQenLG+pn6CHko7ygar
74o4HF1IzqNYCEzr6k4PB1TXSw1wSMTbmIgSbu74WIeAmNDF5P9gQ7NKmYC/KArM
tKdbd2f5NfWax+UDYodp/34H3ajvxBAea8IXv9KZtXIJhMpMkDcDl/Uv
                  </X509Certificate>
                </TrustPoint>
              </CertificateTrustPoint>
            </SignerTrustTrees>
          </SigningCertTrustCondition>
          <TimeStampTrustCondition>
            <TtsCertificateTrustTrees>
              <CertificateTrustPoint>
```

```
                    <TrustPoint>
                      <X509Certificate>
MIIEpjCCA46gAwIBAgIDAYBfMA0GCSqGSIb3DQEBCwUAMEQxCzAJBgNVBAYTAkRF
MRAwDgYDVQQKDAdnZW1hdGlrMSMwIQYDVQQDDBpnZW1hdGlrIExhYm9ydGVzdCBl
R0sgQ0EwHjAeFw0xMTA3MDgwMTIyMjVaFw0xMzA3MDgwMTIyMjVaMIGcMQswCQYD
VQQGEwJERTEgMB4GA1UECgwXZ2VtYXRpayBNdXN0ZXJrYXNzZTFHS1YxEzARBgNV
BAsMClgxMTAxMDMzOTMxEjAQBgNVBAsMCTk5OTU2Nzg5MDEbMBkGA1UEAwwSTGV0
aXppYSBOaWdnZW1leWVyMRMwEQYDVQQEDApOaWdnZW1leWVyMRAwDgYDVQQqDAdM
ZXRpemlhMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqC+s9boZkZcB
3Rq/aqRrXO1RorSpZU+K2ojxn19b3JZJ//RyWweow/5d4NG/kDYyDzbzyZ8pYWNl
+MdQ2Ye4jPvaw12KhUXeQmbW2ZPUeFLrIb9YZ4hL4ABC8w/WWaonrtO1LltjFvYf
UTKR2GacLsuMPidZFx2YLJxbOQ1GQBt5PC/f6d7pqXH3QX22mu4KRBLq82QOZnPY
2Pkcvih1XXzdnwQfSDrgXgxsm02Dn3ZU6/ckRqQcQYXq/L6ZqN1fRUjtBjHQFfXH
DmcS90fF3QXrDjyxKYCtmPvGrvLWX7cfcciPTLMU8nwhNhrGdpH0LWZw679x7PxJ
WO+VZuFAXwIDAQABo4IBRjCCAUIwEwYDVR0lBAwwCgYIKwYBBQUHAwIwEQYDVR0O
BAoECEDLvelPo06yMEQGA1UdIAQ9MDswCQYHKoIUAEwERjAuBgcqghQATARwMCMw
IQYIKwYBBQUHAgIwFRoTb2lkX3BvbGljeV9tdXN0ZV9jcDAwBgUrJAgDAwQnMCUw
IzAhMB8wHTAQDA5WZXJzaWNoZXJ0ZS8tcjAJBgcqghQATAQxMBMGA1UdIwQMMAqA
CE2ldD5PqylIMEMGCCsGAQUFBwEBBDcwNTAzBggrBgEFBQcwAYYnaHR0cDovL29j
c3AuZ2VtYXRpay5kZTo4MDgwL0NNT0NTUC9PQ1NQMA4GA1UdDwEB/wQEAwIHgDA2
BgNVHR8ELzAtMCugKaAnhiVsZGFwOi8vY3JsMDAuZ2VtYXRpay5pbnZhbGlkL25v
dC11c2VkMA0GCSqGSIb3DQEBCwUAA4IBAQCO/FykcCLLU3Q0lWYrOxVROwTTS3sg
M2YDH9/CnDQA1MNRAkJBeE2pwR1Ol/nlZQZJzHPtzuKvL18aj7NMIdqWYJ8Ijx+B
nfoUTNONONksxas0ySnwz2ztLv/3I0/A2VZw2wMXpeNHRj99taI1mLEjaHqdWcSA
L9etm7VCHW3TX4QekLPJfaTAfJhq0koMfouBWjVzFLZTGKQenLG+pn6CHko7ygar
74o4HF1IzqNYCEzr6k4PB1TXSw1wSMTbmIgSbu74WIeAmNDF5P9gQ7NKmYC/KArM
tKdbd2f5NfWax+UDYodp/34H3ajvxBAea8IXv9KZtXIJhMpMkDcDl/Uv
                      </X509Certificate>
                    </TrustPoint>
                  </CertificateTrustPoint>
                </TtsCertificateTrustTrees>
              </TimeStampTrustCondition>
              <AlgorithmConstraintSet>
                <SignerAlgConstraints>
                  <AlgAndLength>
                    <AlgId>http://www.w3.org/2001/04/xmldsig-more#rsa-sha256</AlgId>
                    <MinKeyLength>2048</MinKeyLength>
                  </AlgAndLength>
                  <AlgAndLength>
                    <AlgId>http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256</AlgId
                      >
                    <MinKeyLength>224</MinKeyLength>
                  </AlgAndLength>
                </SignerAlgConstraints>
              </AlgorithmConstraintSet>
            </CommonRules>
          </SignatureValidationPolicy>
        </SignPolicyInfo>
      </SignaturePolicy>
```

B Example SignRequest with Embedded Style Information

```
<dss:SignRequest RequestID="TestRequest" xmlns:dss="urn:oasis:names:tc:dss:1.0
    :core:schema" xmlns:futureid="urn:eu:futureid" xmlns:ds="http://www.w3.org
    /2000/09/xmldsig#">
  <dss:OptionalInputs>
    <dss:SignatureType>urn:ietf:rfc:3275</dss:SignatureType>
    <dss:SignatureForm>urn:oasis:names:tc:dss:1.0:profiles:AdES:forms:BES</
        dss:SignatureForm>
    <dss:SignaturePlacement WhichDocument="testdoc0"><dss:XPathAfter>/doc/
        futureid</dss:XPathAfter></dss:SignaturePlacement>
    <dss:SignedReference WhichDocument="testdoc0">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
            signature"/>
        <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
          <xsl:stylesheet version="1.0" xmlns="http://www.w3.org/1999/XSL/
              Transform"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
            <xsl:output method="xml"/>
            <xsl:template match="/">
              <html xmlns="http://www.w3.org/TR/REC-html40">
                <body>
                  <h2>FutureID Workpackage</h2>
                  <table border="1">
                    <tr bgcolor="#9acd32">
                      <th>Number</th>
                      <th>Name</th>
                    </tr>
                    <xsl:for-each select="doc/futureid/workpackage">
                      <tr>
                        <td><xsl:value-of select="number"/></td>
                        <td><xsl:value-of select="name"/></td>
                      </tr>
                    </xsl:for-each>
                  </table>
                </body>
              </html>
            </xsl:template>
          </xsl:stylesheet>
        </ds:Transform>
      <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></
          ds:Transform></ds:Transforms>
    </dss:SignedReference>
  </dss:OptionalInputs>
  <dss:InputDocuments>
    <dss:Document ID="testdoc0" RefURI="" RefType="DSSReference">
      <dss:InlineXML>
        <doc>
          <futureid>
            <workpackage>
              <number>D33.3</number>
              <name>Implementation of the eSignServices</name>
            </workpackage>
          </futureid>
```

```
        </doc>
      </dss:InlineXML>
    </dss:Document>
  </dss:InputDocuments>
</dss:SignRequest>
```