Libdigidocpp Programmer's Guide

Document Version: 1.3

Library Version: 3.8

Last update: 09.12.2013

1. Document versions

Document information	
Created on	14.03.2013
Reference	Libdigidocpp Programmer's Guide
Receiver	Sertifitseerimiskeskus AS
Author	Kristi Uukkivi
Version	1.3

Version information		
Date	**Version**	**Changes**
14.03.2013	0.1	Initial draft
21.03.2013	0.2	Initial version
26.03.2013	1.0	Revised version
30.05.2013	1.1	Updates according to changes in library's version 3.8.
18.07.2013	1.2	Added information about interoperability testing, updated Libdigidocpp library's implementation notes
09.12.2013	1.3	Updated according to BDOC 2.1 file format's updates (in comparison to BDOC 2.0 format); added Finnish certificates support; updated chapter 5.4, added description of validation statuses and their priorities, including "valid with warnings" status.

Table of contents

2. Introduction

This document describes Libdigidocpp - the C++ library for OpenXAdES/DigiDoc system.

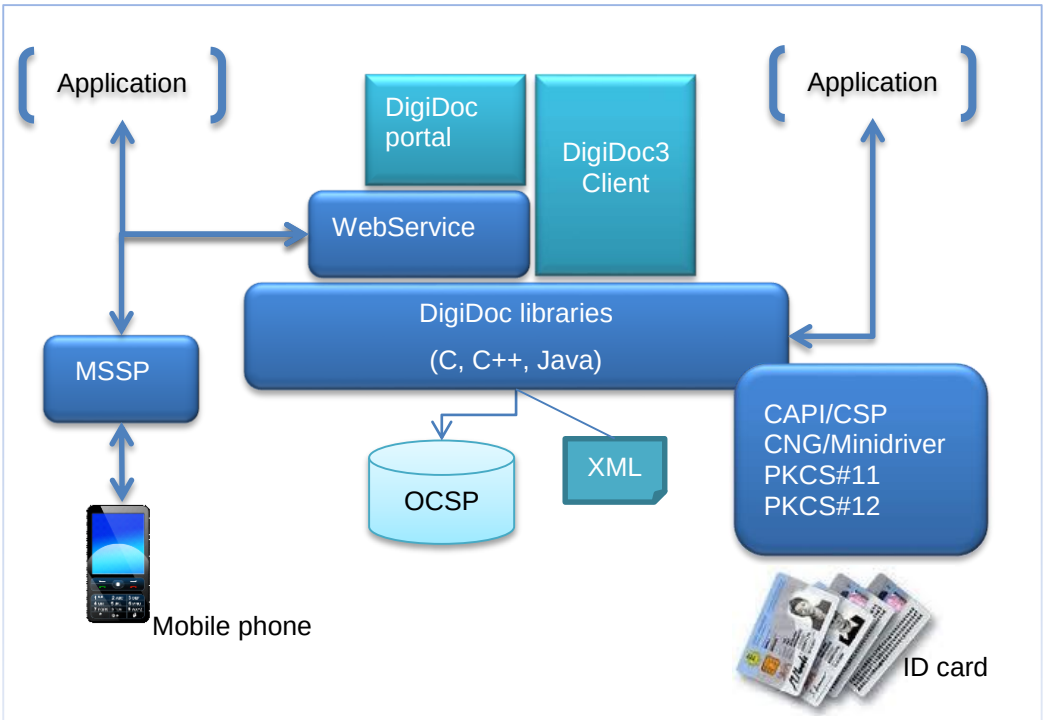
Libdigidocpp is a basic building tool for creating applications handling digital signatures, their creation and verification. The digitally signed files are created in "DigiDoc format" (with .ddoc or .bdoc file extensions), compliant to XML Advanced Electronic Signatures (XAdES), technical standard published by European Telecommunication Standards Institute (ETSI).

This document covers the following information about Libdigidocpp:

- Section 2 introduces the OpenXAdES/DigiDoc framework, its general security model and the main file format to be used for digitally signed files.
- Section 3 gives an overview of the Libdigidocpp library by describing the supported functionality and additional features, the general architecture of components and describes the dependencies.
- Section 4 explains Libdigidocpp's configuration possibilities.
- Section 5 provides samples for handling digitally signed files by using the Libdigidocpp API's classes and methods.
- Section 6 explains using the command line utility program of Libdigidocpp, including sample use cases.
- Section 7 gives overview of supported Estonian and cross-border Certificate Authorities.
- Section 8 describes the interoperability testing of BDOC 2.1 and other DigiDoc file formats.
- Section 9 gives an overview of Libdigidocpp library's implementation notes which provide information about specific features of digitally signed files that are not defined in standards or specification documents but are implemented in Libdigidocpp library.
- Appendix 1 provides a sample digidocpp.conf configuration file.
- Appendix 2 provides a list of XML Schema modifications that have been made to the schemas used in Libdigidocpp library.

2.1 About DigiDoc

Libdigidocpp library forms a part of the wider OpenXAdES/DigiDoc system framework which offers a full-scale architecture for digital signature and documents, consisting of software libraries (C, C++ and Java), web service and end-user applications such as DigiDoc Portal and DigiDoc Client3 according to the following figure:



1. DigiDoc framework

It is easy to integrate DigiDoc components into existing applications in order to allow for creation, handling, forwarding and verification of digital signatures and support file encryption/decryption. All applications share common digitally signed file formats (current versions are BDOC 2.1 and DIGIDOC-XML 1.3).

2.2 DigiDoc security model

The general security model of the DigiDoc and OpenXAdES ideology works by obtaining proof of validity of the signer's X.509 digital certificate issued by a certificate authority (CA) at the time of signature creation.

This proof is obtained in the format of Online Certificate Status Protocol (OCSP) response and stored within the signed document. Furthermore, (hash of the) created signature is sent within the OCSP request and received back within the response. This allows interpreting of the positive OCSP response as "at the time I saw this digitally signed file, corresponding certificate was valid".

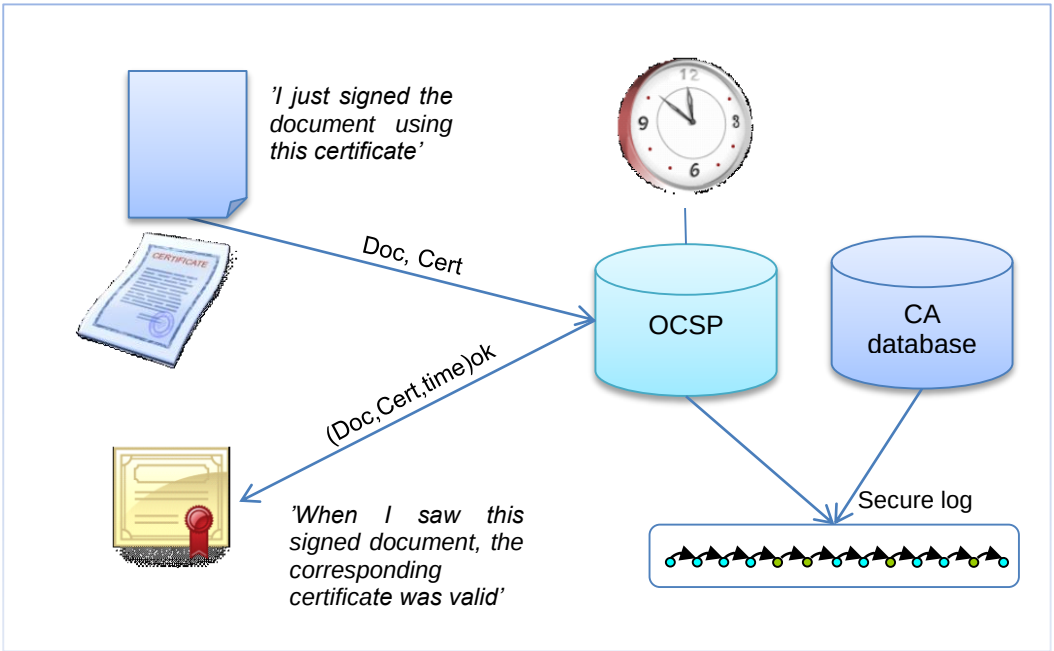
The OCSP service is acting as a digital e-notary confirming signatures created locally with a smart card. From infrastructure side, this security model requires a standard OCSP responder. Hash of the signature is placed on the "nonce" field of the OCSP request structure. In order to achieve the freshest certificate validity information, it is recommended to run the OCSP responder in "real-time" mode meaning that:

- certificate validity information is obtained from live database rather than from CRL (Certificate Revocation List)

- the time value in the OCSP response is actual (as precise as possible)

To achieve long-time validity of digital signatures, a secure log system is employed within the model. All OCSP responses and changes in certificate validity are securely logged to preserve digital signature validity even after private key compromise of CA or OCSP responder. It is important to notice that additional time-stamps are not necessary when employing the security model described:

- time of signing and time of obtaining validity information is indicated in the OCSP response
- the secure log provides for long-time validity without need for archival timestamps



2. DigiDoc security model

2.3 Format of digitally signed file

Actively used digitally signed file formats in DigiDoc system are:

- **BDOC 2.1** - default format for new files in Libdigidocpp library, described in [1];
- **DIGIDOC-XML 1.3 -** see also [2]. **NB!** Libdigidocpp uses CDigiDoc library as a base component to implement support for DIGIDOC-XML 1.3 file format.

DigiDoc system uses file extension **.bdoc** or **.ddoc** to distinguish digitally signed files according to the described file formats. Other historical formats that were used previously are SK-XML, DIGIDOC-XML 1.1, DIGIDOC-XML 1.2 and BDOC 1.0.

The following chapter provides an overview of BDOC 2.1 digitally signed file format which is the preferred format for creating new signed documents in Libdigidocpp library.

The format of the BDOC 2.1 digitally signed file is based on ETSI TS 101 903 standard called **XML Advanced Electronic Signatures (XAdES)** ([5]). The XAdES standard defines formats for advanced electronic signatures that remain valid over long periods of time. The ETSI standard TS 103 171 [10] further profiles the XAdES signature by putting limitations on choices.

The **BDOC Basic Profile (EPES profile)** is an XML structure containing a single cryptographic signature over the well-defined set of data. It does not contain any validation data for full signature validation such as timestamps or certificate validity confirmations. The profile is based on XAdES-EPES (Explicit Policy based Electronic Signature, see [5]).

In order to comply with the security model described in the previous chapter, it is necessary to verify whether the signer's certificate was valid at the (claimed) time of signing. In case of **BDOC with time-marks (TM profile)**, the proof of validity is obtained by using OCSP protocol. The BDOC TM profile is compliant to XAdES LT-Level requirements. The OCSP request's "nonce" field is a DER-encoding of the following ASN.1 data structure:[1]

```
TBSDocumentDigest ::= SEQUENCE {
       algorithm AlgorithmIdentifier,
       digest OCTET STRING
}
```

The element `digest` is a hash value of the binary value of the <SignatureValue> element's contents, element `algorithm` determines the used hash algorithm as defined in RFC 5280 ([9]) clause 4.1.1.2.

Original files (which were signed) along with the signature(s), validation confirmation(s) and certificates are encapsulated within the container. As a result, it is possible to verify signature validity without any additional external information – the verifier should trust the issuer of signer's certificate and the OCSP responder's certificate.

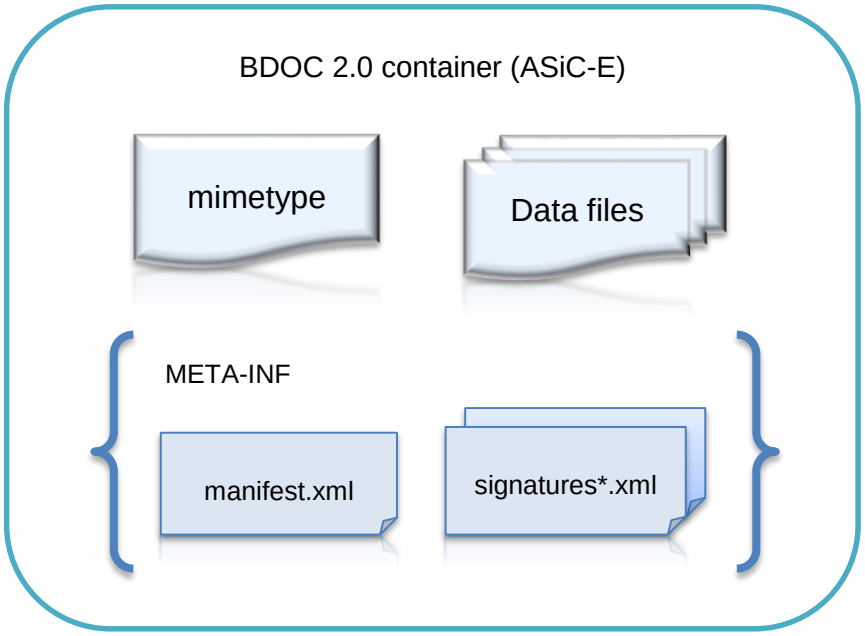
The ETSI standard TS 102 918 [7] called **Associated Signature Containers (ASiC)** defines format of container for encapsulation of signed files and signatures with extra information. The ETSI TS 103 174 [11] profiles in further on. The container type used in case of BDOC 2.1 documents is **Associated Signature Extended form (ASiC-E)**.

ASiC-E container is a ZIP file consisting of the following objects:

- a file named "**mimetype**", containing only the following value: application/vnd.etsi.asic-e+zip
- **data files** in original format.
- **META-INF** subdirectory, consisting of:
  - **manifest.xml** – a file containing list of all folders and files in the container. The list does not contain the "mimetype" file and files in META-INF subdirectory.
  - **signatures*.xml** – one file for each signature, '*' in the file's name denotes the sequence number of a signature (counting starts from zero). The signatures*.xml file also incorporates certificates, validity confirmation and meta-data about the signer.

When BDOC 2.1 container is signed then all files in the container are signed, except of the mimetype file and files in META-INF subdirectory.

[1] Note: OCSP nonce field's value is calculated differently in case of DIGIDOC-XML 1.3 and BDOC 2.1 formats. See the specification documents of these file formats for more information.



3. BDOC 2.1 container's contents

3. Overview

The current chapter gives an overview of **Libdigidocpp** software library by describing the supported functionality, the general architecture of software and hardware components that are involved in signature creation and Libdigidocpp library's dependencies.

3.1 References and additional resources

[1] BDOC2.1:2013	BDOC – Format for Digital Signatures. Version 2.0:2013 https://www.sk.ee/repository/bdoc-spec21.pdf http://id.ee/public/bdoc-spec21-est.pdf
[2] DigiDoc format	DigiDoc file format http://id.ee/public/DigiDoc_format_1.3.pdf
[3] XML-DSIG	IETF RFC 3275: "XML-Signature Syntax and Processing" http://www.ietf.org/rfc/rfc3275.txt
[4] XML-DSIG 1.1	XML Signature Syntax and Processing. Version 1.1 http://www.w3.org/TR/xmldsig-core1/
[5] XAdES	ETSI TS 101 903 V1.4.2 (2010-12) – XML Advanced Electronic Signatures http://www.etsi.org/deliver/etsi_ts/101900_101999/101903/01.04.02_60/ts_101903v010402p.pdf
[6] OpenDocument	OASIS "Open Document Format for Office Applications. Version 1.2 Part 3: Packages" http://docs.oasis-open.org/office/v1.2/cs01/OpenDocument-v1.2-cs01-part3.html#__RefHeading__752803_826425813
[7] ASiC	ETSI TS 102 918 V1.2.1 (2012-02) - Associated Signature Containers http://www.etsi.org/deliver/etsi_ts/102900_102999/102918/01.02.01_60/ts_102918v010201p.pdf
[8] RFC6960	X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP http://tools.ietf.org/html/rfc6960
[9] RFC5280	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile http://tools.ietf.org/html/rfc5280
[10] XAdES Baseline Profile	ETSI TS 103 171 V2.1.1 (2012-03) http://www.etsi.org/deliver/etsi_ts/103100_103199/103171/02.01.01_60/ts_103171v020101p.pdf

[11] ASiC Baseline Profile	ETSI TS 103 174 V2.1.1 (2012-03) http://www.etsi.org/deliver/etsi_ts/103100_103199/103174/02.01.01_60/ts_103174v020101p.pdf
[12] DSA	Estonian Digital Signature Act http://www.legaltext.ee/et/andmebaas/tekst.asp?loc=text&dok=X30081K6&keel=en&pg=1&ptyyp=RT&tyyp=X&query=digitaalallkirja
[13] Release notes	Libdigidocpp library's release notes
[14] CDigiDoc	CDigiDoc Programmer's Guide http://id.ee/public/SK-CDD-PRG-GUIDE.pdf
[15] ETSI TS 102 280 (V1.1.1)	X.509 V3 Certificate Profile for Certificates Issued to Natural Persons http://www.etsi.org/deliver/etsi_ts/102200_102299/102280/01.01.01_60/ts_102280v010101p.pdf
[16] ESTEID profile	Certificates on identity card of Republic of Estonia, version 3.3 https://sk.ee/upload/files/ESTEID_profiil_en-3_3.pdf
[17] Institution certificate profile	Profile of institution certificates and Certificate Revocation Lists, version 1.3 https://sk.ee/upload/files/SK_Profile%20of%20institution%20certificates%20and%20Revocation%20List.pdf
[18] DigiDoc libraries	http://id.ee/index.php?id=30486

3.2 Terms and acronyms

ASiC	Associated Signature Containers
ASiC-E	Extended Associated Signature Containers. A type of ASiC container.
BDOC 2.1 (.bdoc)	Term is used to denote a digitally signed file format which is a profile of XAdES and follows container packaging rules based on OpenDocument and ASiC standards. The document format has been defined in [1], an overview is provided in chapter "2.3 Format of digitally signed file" of the current document.
CRL	Certificate Revocation List, a list of certificates (or more specifically, a list of serial numbers for certificates) that have been revoked, and therefore should not be relied upon.
DIGIDOC-XML (.ddoc)	The term is used to denote a DigiDoc document format that is based on the XAdES standard and is a profile of that standard. The current version is 1.3 which has been described in [2].
ECDSA	Elliptic Curve Digital Signature Algorithm. Digital Signature Algorithm (DSA) which uses elliptic curve cryptography. Used as an alternative to RSA algorithm.

OCSP	Online Certificate Status Protocol, an Internet protocol used for obtaining the revocation status of an X.509 digital certificate
OCSP Responder	OCSP Server, maintains a store of CA-published CRLs and an up-to-date list of valid and invalid certificates. After the OCSP responder receives a validation request (typically an HTTP or HTTPS transmission), the OCSP responder either validates the status of the certificate using its own authentication database or calls upon the OCSP responder that originally issued the certificate to validate the request. After formulating a response, the OCSP responder returns the signed response, and the original certificate is either approved or rejected, based on whether or not the OCSP responder validates the certificate.
SK	AS Sertifitseerimiskeskus (Certification Centre Ltd.). Certificate Authority in Estonia
X.509	an ITU-T standard for a public key infrastructure (PKI) and Privilege Management Infrastructure (PMI) which specifies standard formats for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm
XAdES	XML Advanced Electronic Signatures, a set of extensions to XML-DSIG recommendation making it suitable for advanced electronic signature. Specifies precise profiles of XML-DSIG for use with advanced electronic signature in the meaning of European Union Directive 1999/93/EC.
XML-DSIG	a general framework for digitally signing documents, defines an XML syntax for digital signatures and is defined in the W3C recommendation XML Signature Syntax and Processing

3.3 Supported functional properties

Libdigidocpp is a library of C++ classes offering the functionality of **handling digitally signed files** in supported DigiDoc formats. The following functions are implemented:

- **creating** containers in supported formats and adding data files;
- **signing** DigiDoc documents using smart cards or other supported cryptographic tokens;
- adding **time marks** and **validity confirmations** to digital signatures using OCSP protocol. Note: support for using timestamps is to be implemented in the future.
- **validating** the digital signatures;
- **extracting** data files from DigiDoc document;
- **removing** signatures and data files.

The following table describes additional functional features that are supported with Libdigidocpp.

Feature	Supported values
DigiDoc document format	- **BDOC 2.1 -** the main document format to be used, described in [1]. - **DIGIDOC-XML 1.3 *** – alternative supported file format, described in [2]. Note that using BDOC 2.1 format is preferred for new documents. **Note**: older DigiDoc file formats SK-XML, DIGIDOC-XML 1.1 and DIGIDOC-XML 1.2 are supported only for backward compatibility in case of digital signature verification and data file extraction operations (creating new files and modifying existing files is not supported). **Note**: BDOC 1.0 file format is not supported (files in this format are recognized by the library but handling the files is not supported).
Signature profile	Signature profiles are based on the profiles defined by XAdES ([5]). - **TM -** the default profile, actual certificates and revocation data are added to the signed document to allow verification in future even if their original source is not available; uses time marking. In case of BDOC 2.1 document format, the "SignaturePolicyIdentifier" element is mandatory (see also [1]).
Signature creation module	- **PKCS#11 –** in Linux and OS X environments, the default module for singing with smart card (e.g. Estonian ID card or any other smartcard provided that you have the external native PKCS#11 driver for it). - **CNG API/Minidriver –** Microsoft CNG API and minidriver, the default module for signing with smart card in Windows environment. By default, a dialog window is opened for the user to choose the signing certificate and enter PIN code.
Cryptographic token type**	- **Smart card,** e.g. Estonian ID card. Supported signature creation modules are PKCS#11 and CNG/minidriver.
Public-key algorithm	- **RSA** - **ECDSA -** support for ECDSA algorithm has only been tested with 256 bit keys prime256v1(secp256r1). Testing has been carried out with PKCS#12 software tokens (via PKCS#12 signature creation module in digidoc-tool.cpp utility program).

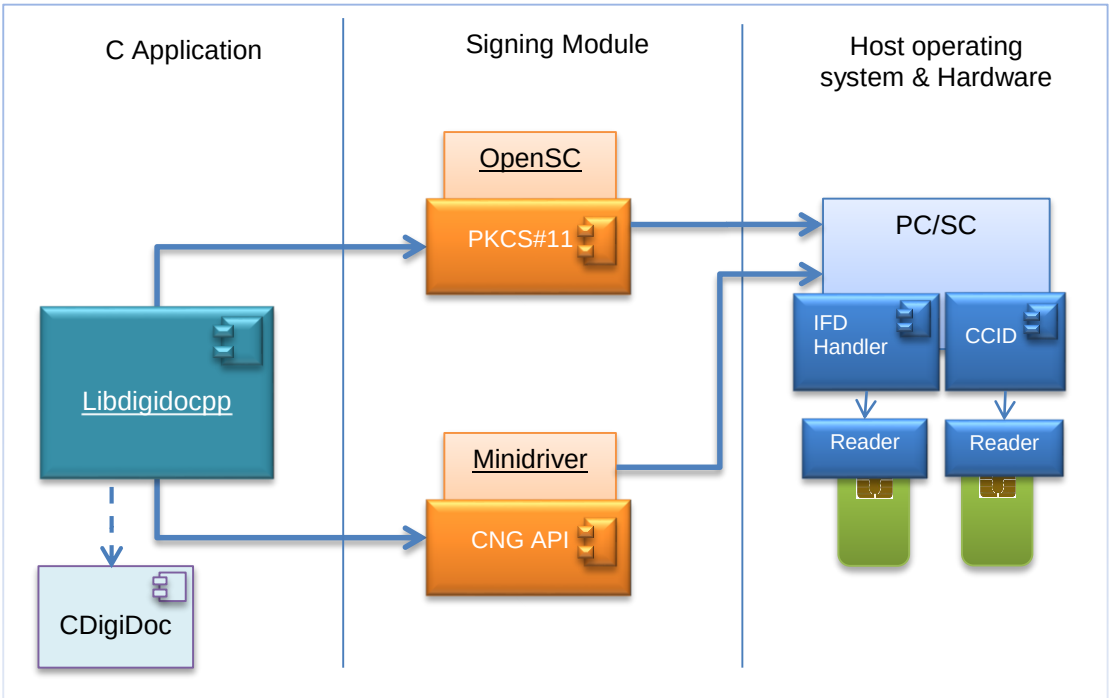
*** Note**: DIGIDOC-XML 1.3 support has been added to Libdicidocpp via CDigiDoc library (the C library of DigiDoc system [14]). Note that the document format is tested in Libdigidocpp indirectly via DigiDoc3 Client desktop application which uses the library as a base layer.

**** Note**: usage of USB cryptostick (Aladdin eToken with Digital Stamp[2] certificate) has been tested indirectly with Libdigidocpp - testing has been carried out via DigiDoc3 Client desktop application which uses Libdigidocpp as a base layer.

[2] http://sk.ee/en/services/Digital-stamp

3.4 Component model

The figure below describes the architecture of software and hardware components that are used when creating signatures with Libdigidocpp library.



4. Components used in Libdigidocpp implementation when signing with smart card

Component	Description
CDigiDoc	C library of DigiDoc system (also known as Libdigidoc, see [14]). Used as a base layer for implementing support for documents in DIGIDOC-XML 1.3 format and enabling some specific operations with older DigiDoc formats (SK-XML, DIGIDOC-XML 1.1, DIGIDOC-XML 1.2).
OpenSC	Set of libraries and utilities to work with smart cards, implementing PKCS#11
PKCS#11	Widely adopted platform-independent API to cryptographic tokens (HSMs and smart cards), a standard management module of the smart card and its certificates
Minidriver	A device driver for controlling interaction with an identity token in Windows operating systems.
CNG API	Microsoft Cryptography API: Next Generation. Programming API for implementing cryptographic functions in Windows environment.
PC/SC	Standard communication interface between the computer and the smart card, a cross-platform API for accessing smart card readers
IFDHandler	Interface Device Handler for CCID readers
CCID	USB driver for Chip/Smart Card Interface Devices
Reader	Device used for communication with a smart card

3.5 Dependencies

3.5.1 Software libraries

Libdigidocpp library depends on the software libraries listed below.

Base Component	Required/ optional	Description
OpenSSL	required	Used for validating certificates and digest values.
XercesC	required	Used in case of BDOC documents: for validating the documents according to XML Schema, reading and writing XML.
XmlSecurityC	required	Used in case of BDOC documents (as an extension for XercesC): for handling signature related components.
XSD	required	Used for dynamically generating C++ source code according to XML Schemas, only used during building process of the library. Required when building the library from source code.
ZLIB	required	Used when generating BDOC files in ZIP format.
Minizip	required	Used when creating ZIP container for BDOC file. If the component is not found from system then bundled version with source code is used. Forms a part of ZLIB component.
CDigiDoc/ Libdigidoc	optional	Used for handling digitally signed files in DIGIDOC-XML format (with .ddoc extension). Libdigidocpp acts as a wrapper for CDigiDoc library. See also [14].
PKCS11	optional	Used for searching for default PKCS#11 driver in the system so that its path could be registered in configuration entries.
Doxygen	optional	Used for generating API documentation from source code.
SWIG	optional	Used for creating C# bindings.

3.5.2 XML Schemas

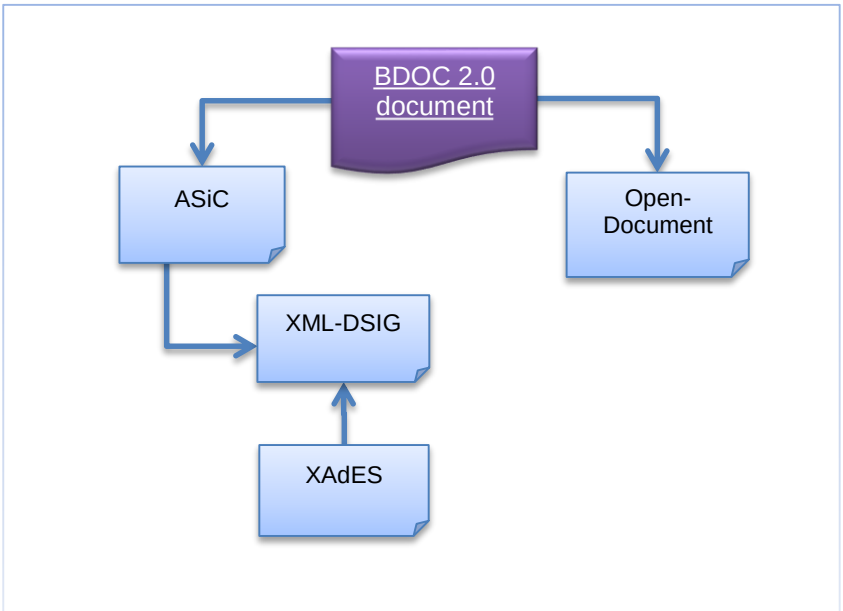
Several XML schemas are used when creating digitally signed documents in **BDOC 2.1** file format and validating their structure. The schemas are included in etc/schema/ subdirectory of the Libdigidocpp distribution package, their description is given in the table below.

Note: some modifications have been made to some of the schemas. Differences in comparison with the original schemas are listed in Appendix 2 of the current document.

Schema file	Description
OpenDocument_manifest.xsd	**OASIS OpenDocument v1.0** ([6]) Defines the structure of META-INF/manifest.xml file in BDOC container. https://www.oasis-open.org/committees/download.php/12570/OpenDocument-manifest-schema-v1.0-os.rng

ts_102918v010201.xsd	**Associated Signature Containers (ASiC)** ([7]) Defines the format of container for encapsulating the signed documents, signatures and additional information. http://www.etsi.org/deliver/etsi_ts/102900_102999/102918/01.02.01_60/
xmldsig-core-schema.xsd	**XML Signature Core Schema Instance (XML-DSIG)** ([3]) Defines XML syntax for digital signatures. http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/xmldsig-core-schema.xsd
XAdES.xsd	**XML Advanced Electronic Signatures (XAdES)** ([5]) Defines a set of extensions to XML-DSIG making it suitable for advanced electronic signature. http://uri.etsi.org/01903/v1.3.2/old/XAdES.xsd
conf.xsd	Configuration properties' schema. Defines the Libdigidocpp configuration file's digidocpp.conf structure (see also "4 Configuring Libdigidocpp"). "

The following figure describes dependencies between the abovementioned schemas (direction of the arrow indicates the direction of dependency).



5. Dependencies between XML Schemas

4. Configuring Libdigidocpp

The following subchapters describe configuration settings for handling BDOC 2.1 files. Information about applying configuration settings for DIGIDOC-XML 1.3 files has been provided in CDigiDoc Programmer's Guide [14].

4.1 Loading configuration settings

Libdigidocpp uses XML configuration file named digidocpp.conf. Configuration file's structure is defined with XML schema "conf.xsd" - the file is included in etc/schema/ subdirectory of Libdigidocpp package. For a sample configuration file, see Appendix 1.

It is possible to use two types of configuration files: **global** and **user's** file. Global file can be used to determine system-wide settings that cannot be altered by a user's file – it can be done separately for each parameter in the file by setting the parameter's "lock" attribute value to "true". User's file can be used to determine user-specific parameter values.

It is possible to use only one configuration file (either global or user's file) or two files in parallel. In the latter case, the matching user file's parameter entries overwrite global file's entries, if the respective parameter is not defined as locked in the global file.

By default, the configuration file's settings are loaded during the library's initialization – Libdigidocpp looks for global and user configuration files from their default locations depending on the environment:

- in case of Windows environment:
  - the global configuration file is looked up from the directory where digidocpp.dll library file is located. If the directory doesn't contain /schema subdirectory then the configuration file is looked up from the current working directory.
  - user configuration file is looked up from the system directory containing application data for the current user: %APPDATA%\digidocpp\digidocpp.conf
- in case of OS X:
  - the global configuration file is looked up from a location in the file system: digidocpp.framework/Resources/digidocpp.conf
  - user configuration file is looked up from $HOME/.digidocpp/digidocpp.conf
- in case of Linux environment:
  - the global configuration file is looked up from a location in the file system: /etc/digidocpp/digidocpp.conf
  - user configuration file is looked up from $HOME/.digidocpp/digidocpp.conf

It is also possible to load global configuration file from a non-default location. In this case, call out the configuration file's initialization method before initializing the library:

```
// Initialize global configuration settings from a non-default location
Conf::init(new XmlConf("<file-path-and-name>"));

// then initialize the library
digidoc::initialize();
```

Local configuration settings can also be set or modified during runtime by calling out the respective set methods of XmlConf class.

4.2 Configuration parameters

Configuration file's elements and their attribute names are defined in conf.xsd file.

Below is a description of the configuration file's parameters. The attribute "lock", when set to "true" can optionally be used to determine parameter values which should not be overwritten by another configuration file (e.g. when using global and user's configuration files in parallel; see also the previous section for more information).

Logging settings

Parameter name	Comments
log.file	Location of the log file where the logging output is written, e.g. ***/tmp/digidocpp.log*** or ***C:\Temp\digidocpp.log*** If left unspecified then the logging output is written to standard output stream.
log.level	Used for controlling the level of detail of the logging output messages, higher number value indicates higher level of detail. Possible values are: ***1*** – error messages, ***2*** – warning messages, ***3*** – info messages, ***4*** – debug messages.

PKCS#11 settings

Parameter name	Comments
pkcs11.driver.path	PKCS#11 driver library to be used when communicating with the smart card. With Estonian ID cards for example, the following PKCS#11 libraries are used: ***opensc-pkcs11.so*** (used in Linux environment) ***opensc-pkcs11.dll*** (used in Windows environment)

CA certificate settings

The CA certificates are used to check if the signer's certificate has been issued by a trusted CA.

This parameter enables determining a directory in file system containing trusted CA certificates (in PEM format). If left unspecified then the CA certificates are loaded from default locations:

- in case of Windows environment - the native Windows certificate store;
- OS X environment - digidocpp.framework/Resources/certs;
- Linux environment - /etc/digidocpp/certs.

Libdigidocpp supports Estonian and Finnish CA certificates. The Estonian live CA and OCSP certificate files have been included in the Libdigidocpp distribution; Finnish live certificates have to be installed with a separate package, accessible from https://installer.id.ee/media/windows/Eesti_ID_kaart_finsertifikaadid.msi

Both Estonian and Finnish test certificates have to be installed separately and can be accessed from https://installer.id.ee/media/windows/Eesti_ID_kaart_testsertifikaadid.msi.

More information about the supported CA-s and certificates can be found from chapter "7 National ".

SK issued live CA and OCSP certificate files are included in the Libdigidocpp distribution but the test certificate files are not. In order to use the test certificates, you need to install them separately (the installation package is accessible from https://installer.id.ee/media/windows/Eesti_ID_kaart_testsertifikaadid.msi).

Note: test certificates should not be used in live applications as the Libdigidocpp library does not give notifications to the user in case of test signatures. In case of live applications, the test certificates should be removed.

Parameter name	Comments
cert.store.path	Directory in file system which contains trusted CA certificates, e.g. ***C:\certs***

XML Schema settings

Parameter name	Comments
xsd.path	Path to the XML schemas that are used for validating DigiDoc files' structure, e.g. ***C:\schemas*** See also section "3.5.2 XML Schemas" for more information.

HTTP proxy settings

The following settings need to be specified only if using a proxy to access internet.

Parameter name	Comments
proxy.host	Specifies the proxy hostname, e.g. ***proxy.example.net***
proxy.port	Specifies the proxy port, e.g. ***8080***
proxy.user	Specifies the proxy username.
proxy.pass	Specifies the proxy password.

Settings for signing OCSP requests

Whether you need to sign the OCSP requests sent to your OCSP responder or not depends on your responder.

Some OCSP servers require that the OCSP request is signed. To sign the OCSP request, you need to obtain and specify the certificates, which will be used for signing.

For example, accessing the SK's OCSP Responder service by private persons requires the requests to signed (access certificates can be obtained through registering for the service[3]) whereas in case of companies/services, signing the request is not required if having a contract with SK and accessing the service from specific IP address(es). It is not necessary to sign OCSP requests in case of using OCSP test-responder (see the next sub-section for more information).

By default, the parameter "pkcs12.disable" value is set to "true" – i.e. the OCSP requests will not be signed. If setting this to "false", you will also need to provide your access certificate file's location and password that have been issued to you for this purpose.

Parameter name	Comments
pkcs12.disable	Specifies if the OCSP requests are signed or not. Possible values are: ***true*** – OCSP requests are not signed; ***false*** – OCSP requests are signed.
pkcs12.cert	Specifies your access certificate's PKCS#12 container's filename, e.g. ***./home/132936.p12d***
pkcs12.pass	Specifies your access certificate's PKCS#12 container's password, e.g. ***m15eTGpA***

[3] https://sk.ee/getaccess/index.php?lang=eng

OCSP responder settings

OCSP responder settings are used during signature creation, when adding OCSP confirmation to a signature and during signature validation, when validating the signer's certificate and OCSP response.

The default address provided (http://ocsp.sk.ee) is for the real-life OCSP Responder address to be used for Estonian ID cards.

The OpenXAdES OCSP Responder address (http://www.openxades.org/cgi-bin/ocsp.cgi) can be used for testing purposes. For more information on using the OpenXAdES testing environment, please refer to http://www.id.ee/?lang=en&id=35755.

Note that in case of Finnish live ID-cards, OCSP confirmation is acquired from proxy OCSP service (http://sk.ee/en/services/validity-confirmation-services/proxy-ocsp/). The OCSP responder server URL value (http://ocsp.sk.ee/_proxy) has been written directly into the source code and cannot be configured.

OCSP parameter name	Comments
ocsp issuer	The "issuer" parameter's name stands for OCSP responder certificate issuer's Common Name (CN) value, e.g. ***ESTEID-SK 2007.*** The element's value specifies OCSP responder server's URL address. There are separate URLs for accessing SK's live and test OCSP responder services: ***http://ocsp.sk.ee*** - live environment ***http://www.openxades.org/cgi-bin/ocsp.cgi*** - testing environment

5. Using Libdigidocpp API

Libdigidocpp library offers creating, signing and verification of digitally signed documents, according to XAdES [5] and XML-DSIG [3] standards. In next chapters a short introduction is given on the main API calls used to accomplish the above mentioned.

For additional information about the classes and methods described in the following subchapters, see the full API description (created with Doxygen) that is included in the Libdigidocpp library's distribution package.

See also chapter "3.3 Supported functional properties" for an overview of supported document formats, signature profiles, signing modules, etc.

Note that Libdigidocpp uses internal memory buffers in case of all the operations, so that intermediary data is not written to temporary files on the disk. Also, the data files to be added to a DigiDoc container can be read from a data stream and later extracted from the container to a stream so that the data can be kept in memory.

5.1 Initialization

Libdigidocpp's initialization method initializes dependent libraries (see also chap. "3.5.1 Software libraries"), loads configuration settings from default configuration files and initializes certificate store according to default settings (either the system's native certificate store or a directory containing the trusted certificates). The default configuration settings are described in chap. "4.1 Loading configuration settings".

If you would like to use non-default settings then call out the configuration file's initialization before initializing the library, for example:

```
using namespace digidoc;

// Optionally initialize global configuration file to use non-default settings
Conf::init(new XmlConf("<file-path-and-name>"));

// Initialize the library
digidoc::initialize();
```

5.2 Creating and signing a DigiDoc document

5.2.1 Creating a DigiDoc container

Create a new container object and specify the DigiDoc document's type, for example:

```
Container doc(Container::AsicType);
```

Container class is used to incorporate the data of a DigiDoc document. The supported container types are defined in Container.h, with the DocumentType enumeration:

- **AsicType** – creates new **BDOC 2.1** document with mime-type "application/vnd.etsi.asic-e+zip". The document format is described in chapter "2.3 Format of digitally signed file".
- **DDocType** – creates new **DIGIDOC-XML 1.3** document. Note that using BDOC 2.1 format is preferred for new documents. Support for DIGIDOC-XML 1.3 format has been added to Libdigidocpp via CDigiDoc library [14]. Note that usage of this document format is tested only indirectly via DigiDoc3 Client desktop application which uses CDigiDoc as a base layer.

Note: the functionality of creating new files in DigiDoc file formats SK-XML, DIGIDOC-XML 1.1, DIGIDOC-XML 1.2 and BDOC 1.0 is not supported.

5.2.2 Adding data files

Data files can be added to a DigiDoc container in two alternative ways:

1. adding the data from an input stream (i.e. the data file contents can be read from internal memory buffer):

```
void Container::addDataFile(std::istream *is, // input stream
       const std::string &fileName, // file name that is written to
       // the container
       const std::string &mediaType); // mime type of the data file
```

2. adding the data by reading the it from file system

```
void Container::addDataFile(const std::string &path, //data file's name and
                // path in file system
       const std::string &mediaType); // mime type of the data file
```

Parameter mediaType in the methods above stands for a MIME type of the data file, for example "text/plain" or "application/msword". Value "application/octet-stream" is used by default.

Calling out any of the methods listed above shall create a new DataFile object and add it to the DigiDoc container's data file collection.

Note that in order to add a data file to a container, the container has to be unsigned and there shouldn't be an existing data file with the same name in the container. If a container is signed then it is possible to add data files to it only after the signatures are removed.

5.2.3 Adding signatures

It is possible to add a signature to a container only if it contains at least one data file, multiple signatures can be added to a single container. The signer's certificate[4] and PIN code to access the private signature key are required during signing.

Signing can be done by using different modules for accessing the signature token:

- PKCS#11 module - the default module for singing with smart card (e.g. Estonian ID card or any other smartcard provided that you have the external native language PKCS#11 driver for it).
- Microsoft CNG API and minidriver for signing with smart card in Windows environment. By default, a dialog window is opened for the user to choose the signing certificate and enter PIN code.

To start adding signatures, firstly declare pointer to the Signature object to be created:

```
Signature *signature = 0;
```

Next, we create the signature and its OCSP confirmation and add them to the DigiDoc container. Note that the OCSP responder settings should be configured before calling out the methods in the following examples (see also chap. "4.2 Configuration parameters", under "OCSP responder settings").

Create the signature according to the module that is used for accessing the signing token as described in the following sections.

5.2.3.1 Signing with PKCS#11 module

```
PKCS11Signer *signer = new PKCS11Signer("<PKCS11-driver-path>"); // optionally
// specify PKCS#11 driver's location. If left unspecified then location is
```

[4]Signing can be done with a certificate that has "Non-Repudiation" value set in its "Key Usage" extensions field. See also chap. "9 Libdigidocpp implementation notes".

```
// looked up from configuration file's parameter "pkcs11.driver.path".

signer->setPin("<pin-code>"); //PIN2 in case of Estonian ID cards
```

Optionally specify additional signer's data:

```
std::string city, state, postalCode, country;
std::vector<std::string> roles;
signer->setSignatureProductionPlace(city, state,
    postalCode, country); // location where the signature is created
signer->setSignerRoles(roles); // role(s) of the signer
```

Create the signature:

```
signature = doc.sign(signer);
```

If you would like to add PIN insertion dialog window for the signer to enter the PIN code then you can write a new class which extends the PKCS11Signer class, overwrite the std::string pin(const X509Cert &cert) method and write your own PIN dialog implementation code there.

5.2.3.2 Signing via Microsoft CNG API (in Windows environment)

```
// Variables for optional signer's data
std::string city, state, postalCode, country, pin;
std::vector<std::string> roles;
bool selectFirst;

signature = doc.sign(
    city, state, postalCode, country, // optionally set the signing location
    roles, // optionally specify the signer's role(s)
    pin,
    selectFirst);
```

By default, when signing via CNG API then a dialog window is displayed to the user for choosing the signing certificate and inserting PIN code. However, the default behaviour can be changed with parameters "pin" and "selectFirst" of the above mentioned method:

- if input parameter "**pin**" is specified when calling out the method then the specified PIN value is used for signing and dialog window for PIN insertion is not displayed to the user.
- boolean parameter "**selectFirst**", if set to "true", determines that the first signing certificate that is found from the certificate store is chosen for signature creation and the certificate selection's dialog window is not displayed to the user.

5.2.3.3 Validating the created signatures

After the signature has been added to the container, it should be validated before writing the signed container to an output file. For validating the signature, do as follows:

```
signature->validate();
```

The validation method above validates the signed data files', signer certificate's and OCSP confirmation's correspondence to the signature value. Note that the validation method above does not validate other signatures which may belong to the same container.

5.3 Reading and writing DigiDoc documents

In order to read an existing DigiDoc file from the file system, do as follows:

```
Container doc("<input-file's-path>");
```

The method above reads in the DigiDoc file from the specified location in file system and creates the respective Container object representing the document's data. The file's structure

is also validated during its parsing according to the corresponding standards (see also [2] for DIGIDOC XML 1.3 documents and [1] for BDOC 2.1 documents).

Write a DigiDoc file (represented with a Container object) to file system with the following method:

```
doc.save("<output-file's-path>");
```

5.4 Validating signatures

Validation of a signed DigiDoc document consists of three main steps:

1. Call out the main validation method of the library. If there are multiple validation errors then get the errors list.
2. Check for additional errors/warnings (separate implementation);
3. Determine the validation status of the document (according to the returned error codes and validation status priorities).

5.4.1 Using the main validation method

You can validate a signature and its OCSP confirmation with method:

```
void Signature::validate();
```

If an exception is thrown from the validation method then the signature can be either INVALID or VALID WITH WARNINGS; otherwise the signature is VALID. Before determining the final validation status, additional errors must be checked, as described in the following chapters.

If an exception is thrown then its causes can be retrieved with the following method:

```
vector<Exception> Exception::causes();
```

5.4.2 Checking for additional errors/warnings

There are validation cases that are not checked in the default validation method of the library, instead, separate methods for checking the specific situations have to be implemented by the library's user. In Libdigidocpp library, checking for a **test signature** and **old file format** must be done separately.

The following subchapters describe how these checks can be implemented. After checking for additional errors/warnings, collect all of the error codes and continue with determining the validation status as described in the next chapter.

5.4.2.1 Checking for test signature

Test signature is a signature that has been created by using test certificates (e.g. signer's certificate and/or OCSP responder server's certificate have been issued for testing purposes).

Sample code for checking for test signature can be found from digidoc-tool.cpp utility program, method:

```
open(int argc, char* argv[]); //utility program's command "open"
```

For identifying if a certificate is a SK issued test certificate, you can use the following method as a sample code:

```
type(const X509Cert &cert, bool ocsp);
```

The identification is done with comparing certificate policy OID values.

5.4.2.2 Checking for old file formats

Sample for checking old versions can be found from two sources.

1. CDigiDoc library's utility program's source code in cdigidoc.c source file, method

```
cmdReadDigiDoc(SignedDoc** ppSigDoc, DEncEncryptedData** ppEncData, int
nMode); //utility program's command -in
```

The check for old file formats is implemented in method:

```
checkOldFormatVer(SignedDoc* pSigDoc);
```

2. It is possible to check the source code of DigiDoc3 Client desktop application, accessible from https://svn.eesti.ee/projektid/idkaart_public/trunk/qdigidoc/.

5.4.3 Determining the validation status

After validating the signed DigiDoc document, the validation result must be determined by the library's user. Final validation result must be one of the possible validation statuses that are described in the table below, the status must be chosen according to its priority.

The validation status priorities have to be applied in two cases:

1. **Returning a validation result of a single signature:**
   If there are more than one validation errors that occur when validating a single signature in DigiDoc container then the overall status of the signature should be chosen according to the status priorities.

2. **Returning a validation result of the whole DigiDoc container:**
   If there are more than one signatures in a DigiDoc container and the signatures have different validation statuses or validation of the container structure returns a different status then the overall status of the DigiDoc file should be chosen according to the status priorities.

NB! User of the library has to determine the validation status according to the error code that is returned by the library's validation method.

Priority	Status	Error code	Description
1	**INDETERMINATE/UNKNOWN**	**10** CertificateIssuerMissing (signer's certificate is unknown) **6** CertificateUnknown (OCSP responder certificate is unknown)	Validation process determines that one or more of the certificates included in the document are unknown or not trusted, i.e. the certificates have been issued by an unknown Certificate Authority (the CA has not been added to trusted list). **Notes**: • The file and signature(s) are not legally valid. • If the CA will later be added to the trusted list/trust store then the validation status can change to any of the other statuses described in the current table. **Suggested warning message** (also displayed in DigiDoc3 Client): "Signature status is displayed as unknown if you don't have all validity confirmation service certificates and/or certificate authority certificates installed into your computer" **More info**: http://www.id.ee/index.php?id=35941 **Sample file**: SS-4_teadmataCA.4.asice

Priority	Status	Error code	Description
2	**INVALID**	All errors except of the ones that are regarded as warnings by the library's user.	Validation process returns error(s), the errors have not been explicitly determined as minor error(s) by the library's user. **Note:** • The file and signature(s) are not legally valid. • No further alterations should be made to the file, i.e. no signatures should be added or removed.
3	**TEST**	N/A (Separate error code has not been determined. See also chap. 5.4.2.1)	Test certificates have been used in the signed file (e.g. signer's certificate and/or OCSP responder server's certificate have been issued in testing purposes). **Notes**: • Test signature is not legally binding even if the signature is valid. • This status is used in combination with the other validation statuses described in the current table. **Suggested warning message** (also displayed in DigiDoc3 Client): "Test signature" **More info**: http://www.id.ee/index.php?id=30494 **Sample file**: aladdin3.6.ddoc
4	**VALID WITH WARNINGS**	See the next section.	Validation process returns error(s) that have been previously explicitly categorized (by the library's user) as minor technical errors. Note that this status is used only in exceptional cases, more details of which are given in the next chapter. **Notes**: • The file and signature(s) are handled as legally valid. • The error(s) are regarded as validation warnings. • Validation warnings should be displayed to the user. • No further alterations should be made to the file, i.e. no signatures should be added or removed. • Creator of the file should be informed about the error situation.
5	**VALID**	N/A	Validation process returns no errors. The signature is legally valid.

The error codes described in the table above are defined in Exception.h source file.

Sample code of DigiDoc file validation can be found from digidoc-tool.cpp utility program, from the following method:

```
open(int argc, char* argv[]); //utility program's command "open"
```

5.4.3.1 Validation status VALID WITH WARNINGS

In special cases, validation errors can be regarded as minor technical errors and the file's validation status can be regarded as VALID WITH WARNINGS instead.

NB! User of the DigiDoc library has to decide on his/her own when to use VALID WITH WARNINGS status instead of INVALID: there may be different interpretations of the severity of validation errors in different information systems then the final decision when to use this

status has to be made by the library's user according to the requirements of the specific information system.

It is recommended to use the validation status VALID WITH WARNINGS in case of the error situations that are included in the table below - these error situations are regarded as VALID WITH WARNINGS in DigiDoc applications and software libraries, including:

- DigiDoc3 Client desktop application,
- JDigiDoc, Libdigidocpp and CDigiDoc software libraries' utility programs.

Table 1. Validation error codes recommended to be handled as VALID WITH WARNINGS

Status	Error code	Related DigiDoc file format	Description
VALID WITH WARNINGS	**12** ReferenceDigestWeak **13** SignatureDigestWeak	BDOC 2.1	Weaker digest method (SHA-1) has been used than recommended when calculating either <Reference> or <Signature> element's digest value. Suggested warning message (also displayed in DigiDoc3 Client): "The current BDOC container uses weaker encryption method than officially accepted in Estonia." **Sample file**: 23147_weak-warning-sha1.bdoc
	14 DataFileNameSpaceWarning	DDOC 1.0 DDOC 1.1 DDOC 1.2 DDOC 1.3	<DataFile> element's xmlns attribute is missing. **Suggested warning message** (also displayed in DigiDoc3 Client): "This DigiDoc documents has not been created according to specification, but the digital signatures is legally valid. You are not allowed to add or remove signatures to this container." **More info**: http://www.id.ee/?id=36213 **Sample file**: 18912.ddoc
	15 IssuerNameSpaceWarning	DDOC 1.1 DDOC 1.2 DDOC 1.3	<IssuerSerial><X509IssuerName> and/or <IssuerSerial><X509SerialNumber> element's xmlns attribute is missing. **Suggested warning message** (also displayed in DigiDoc3 Client): "This DigiDoc documents has not been created according to specification, but the digital signatures is legally valid. You are not allowed to add or remove signatures to this container." **More info**: http://www.id.ee/?id=36213 **Sample file**: 20bait_nonce.ddoc

Status	Error code	Related DigiDoc file format	Description
	N/A (Separate error code has not been determined. See also chap. 5.4.2.2)	DDOC 1.0 DDOC 1.1 DDOC 1.2	DigiDoc file's version is older than currently supported. Note that the error situation affects only the container and not the signatures, therefore, in DigiDoc libraries, it is returned and displayed only at container level. **Suggested warning message** (also displayed in DigiDoc3 Client): "The current file is a DigiDoc container that is not supported officially any longer. You are not allowed to add or remove signatures to this container" **More info**: http://www.id.ee/index.php?id=36161 **Sample file**: DigiDoc 1.0 (tartu_ja_tallinna_koostooleping).ddoc

Sample code for determining validation warnings can be found from digidoc-tool.cpp utility program, see command open, function validate().

5.4.4 Additional information about validation

5.4.4.1 Overview of validation activities

Overview of validation activities is as follows:

1. checking that all the data files and signature's meta-data (signer's role, etc.) are included in the signature by calculating the data objects' digest values and comparing them with the <Reference> element values in the signature;
2. checking that the claimed signer's certificate is the actual certificate that was used for signing; checking that the "Non-repudiaton" value is set in the "Key Usage" extension of the signer's certificate;
3. checking that the signature value is correct by decrypting the value with the signer's public key and comparing the result with digest calculated from <SignedInfo> element block;
4. checking that the OCSP response confirms the signer certificate's validity and corresponds to the signature value (by comparing the digest value of <SignatureValue> element's value and OCSP response's nonce value);
5. checking that the signer's and OCSP responder's certificates are trusted (i.e. the certificates' issuers are registered in trust store).

5.5 Extracting data files

A data file can be extracted from container and written to the specified location in the file system or to an output stream.

1. You can write the data file to a stream and keep it in memory:

```
void DataFile::saveAs(std::ostream &os);
```

2. The file can be written to file system with the following method:

```
void DataFile::saveAs(const std::string& path);
```

List of all the document's data files can be retrieved with the following method:

```
DataFileList Container::dataFiles();
```

For example, read in a DigiDoc document and write its data files to file system as follows:

```
Container doc("<input-file's-path>"); // read in a document
DataFileList files = doc.dataFiles(); // get the data files' list

for(DataFileList::const_iterator i = files.begin(); i != files.end(); ++i){
   try {
      std::string dst = i->fileName(); // get the data file's name
      i->saveAs(dst); // save the data file to working directory
   } catch(const Exception &e) {
      printf("  Document %s extraction: FAILED\n", i->fileName().c_str());
   }
}
```

5.6 Removing signatures and data files

In order to remove a signature from DigiDoc document, use the following method:

```
void Container::removeSignature(unsigned int id);
```

Data files can be removed from a container only after all of its signatures have been removed. Use the following method to remove a data file from DigiDoc container:

```
void Container::removeDataFile(unsigned int id);
```

"Id" parameters of the abovementioned methods represent the signature's and data file's sequence numbers in the container. The identifiers are determined when a data file or signature is added to the container, counting starts from zero.

Note: the functionality of modifying files in DigiDoc file formats SK-XML, DIGIDOC-XML 1.1, DIGIDOC-XML 1.2 and BDOC 1.0 is not supported.

5.7 Shutting down the library

After finishing work with Libdigidocpp, then the last task is to shut down the library:

```
digidoc::terminate();
```

The termination method closes libraries used in Libdigidocpp implementation and deletes temporary files that may have been written to disk when working with the library.

5.8 Exception handling

The Libdigidocpp library may throw exceptions that are instances of Exception class (defined in Exception.h source file). The code which uses Libdigidocpp's API should be wrapped in a try/catch block as follows:

```
try {

       // code implementation

} catch(const Exception &e) {
    printf("Exception:%s\n", parseException(e).c_str()); // Sample exception
                                                         // handling method
}
```

An Exception instance thrown by the library may contain a stack trace of the hierarchy of exceptions. For example, to parse the whole stack trace, do as follows:

```
std::string parseException(const Exception &e)
{
    std::string result = e.msg() + "\n"; // the error message is retrieved
    Exception::Causes list = e.causes(); // list of exceptions in lower level
    // Iteration through the list of causes:
    for(Exception::Causes::const_iterator i = list.begin(); i != list.end(); ++i)
        result += parseException(*i); // Parsing the exceptions recursively
    return result; // The error message is returned
}
```

6. Libdigidocpp utility program

The command line utility program digidoc-tool.exe which is included in the Libdigidocpp distribution can be used to test the library or simply use it directly to handle digitally signed documents.

The general format for executing the program is:

```
> digidoc-tool [command] [options] [input/output file]
```

6.1 Creating and signing a document

Command "create" can be used to create a new DigiDoc container, add data files, optionally some meta-info about the signer and sign the document. Documents can be created only in **BDOC 2.1** format.

General form of the command is:

```
> digidoc-tool create --file=<data-file> <output-bdoc-file>
```

Available options:

--file=	Required Data file(s) to be signed. The option can occur multiple times.
--pin=	Optional If PIN is not provided with this parameter value and (the default) PKCS#11 module is used for signing then the utility program asks for the user to insert PIN code to command line during the program's execution time.
--city=	Optional City where the signature is created.
--state=	Optional State or province where the signature is created.
--postalCode=	Optional Postal code of the place where the signature is created.
--country=	Optional Country of origin. ISO 3166-type 2-character country codes are used (e.g. EE)
--role=	Optional Signer's role(s). The option can occur multiple times.
--sha(1,224, 256,384,512)	Optional Used for testing purposes. Specifies the hash function that is used for calculating digest values. If not specified then SHA-256 is used by default.

Options for specifying module used for accessing the signing token - possible alternatives are PKCS#11 and CryptoAPI/CNG). When signing module is not specified then PKCS#11 module is used by default.

--pkcs11[=]	Optional Signing is done via PKCS#11 module - the default module for singing with smart card in Linux and OS X. When signing via PKCS#11 module then the parameter's value can be used to specify the path and filename of PKCS#11 driver in your file system. For example, "opensc-pkcs11.dll" in Windows environment and "opensc-pkcs11.so" in Linux. If the parameter's value is left unspecified then PKCS#11 driver's location is looked up from configuration file (see also chap. "4.2 Configuration parameters").
--cng	Optional Set the parameter to sign via Microsoft CNG API (in Windows environment). If "--pin" parameter's value is not set then PIN insertion dialog is displayed to the user. Parameter "--cng" may optionally be used along with parameter "--selectFirst".
--selectFirst	Optional Additional parameter that can optionally be used along with parameter "–cng". When the parameter is set then the first certificate in Windows certificate store is chosen for signature creation. If the parameter is not set then certificate selection dialog window is displayed to user.

Note: there is also PKCS#12 signing module support implemented in Libdigidocpp utility program which is used for testing signature creation with ECC keys. More detailed description about usage scenarios is to be determined in the future.

Sample commands for creating and signing DigiDoc files:

Sample: creating new BDOC file, adding multiple data files and signing via PKCS#11 driver

```
> digidoc-tool create --file=file1.txt --file=file2.pdf --country=Estonia
--state=Harjumaa --city=Tallinn --postalCode=12345 --pkcs11 demo-container.bdoc

Input:
  - file1.txt               - a data file to be added to container
  - file2.pdf               - a data file to be added to container
  - Estonia                 - country where the signature is created
  - Harjumaa                - county where the signature is created
  - Tallinn                 - city where the signature is created
  - 12345                   - postal code of the signature creation location
  - --pkcs11                - signing is done via PKCS#11 module
  - demo-container.bdoc     - container to be created (in BDOC2.0 format)
```

Sample: creating new BDOC file, adding data file and signing via CNG API

```
> digidoc-tool create --file=file1.txt --cng demo-container.bdoc

Input:
  - file1.txt               - a data file to be added to container
  - --cng                   - CNG API is used for signing
  - demo-container.bdoc     - container to be created (in BDOC2.0 format)
```

Sample: creating new BDOC file, adding data file and signing via CNG API, dialog windows for certificate selection and PIN insertion are not displayed

```
> digidoc-tool create --file=file1.txt --cng --selectFirst --pin=01497 demo-
container.bdoc

Input:
  - file1.txt               - a data file to be added to container
```

```
- --cng                    - CNG API is used for signing
- --selectFirst            - the first signing certificate in store is used for
                           signing
- 01497                    - PIN code (PIN2 in case of Estonian ID cards)
- demo-container.bdoc      - container to be created (in BDOC2.0 format)
```

6.2 Opening document, validating signatures and extracting data files

Command "open" enables to read in an existing DigiDoc document, print out a list of its contents and validate signatures. By specifying the additional option --extractaAll, then the data files are extracted from the container and stored on the disk. All DigiDoc file formats are supported with this command (except of BDOC1.0).

General form of the command is:

```
> digidoc-tool open <input-container-file>
```

Available options:

--extractAll	Optional If set, then all of the input container's data files are extracted and written to disk. If an output directory is not specified with the value of this parameter then the extracted files are written to the same directory where the input file is located.
--warnings= (ignore, warnings, error)	Optional Enables to choose the displaying of validation warnings (if present) of the file being opened. Can be used to test the warnings system of the utility program (see also "Validation status VALID WITH WARNINGS"). The options include: - warnings – the default value used. The minor technical errors that are considered as warnings, are printed out as warnings. - error – the errors that are otherwise considered as warnings (by the utility program), are printed out as errors. - ignore – the errors that are otherwise considered as warnings (by the utility program), are not printed out. If there are any other errors present then these are treated as usual.

Output of the default command contains the following data of the container:

```
Container type: <container's mime-type>
Documents (<number of data files in container>):
  Document (<data file's mime-type>): <file's name> (<file's size> bytes)
Signatures (<number of signatures in container>):
  Signature <signature's sequence number> (<signature's profile>):
    Validation: <signature validation result: OK/FAILED>
    EPES policy: urn:oid: <signature policy identifier OID>
    SPUri: <URL to the BDOC 2.1 specification document>
    Signature method: <signature method URI>
    Signing time: <signing time according to computer's settings (not the
                  official signing time)>
```

```
      Signing cert: <subject CN field's value>
      Produced At: <time of OCSP response's issuance, i.e. official signing time>
      OCSP Responder: <OCSP responder certificate CN field's value>
      OCSP Nonce (<length in bytes>): <OCSP responses nonce field's value (has to
                  correspond to the <SignatureValue> element's hash)>
      Warnings: <possible validation related warnings (see explanation below)>
```

Note: by default, if the signature validation process discovered errors that are regarded as minor technical errors in digidoc-tool.cpp utility program then the document is considered as VALID WITH WARNINGS, the errors are printed out as warnings to the end user. See also chapter "5.4.3 Determining the validation status".

Sample commands for validating signatures and extracting data files:

```
Sample: opening BDOC container, listing its contents and validating signatures
> digidoc-tool open demo-container.bdoc

Input:
  - demo-container.bdoc      - input DigiDoc file which contents are listed and
                             signatures validated
Output:
  Container type: application/vnd.etsi.asic-e+zip
  Documents (2):
    Document (application/octet-stream): file1.txt (434 bytes)
    Document (application/octet-stream): file2.pdf (476841 bytes)
  Signatures (1):
    Signature 0 (EPES/time-mark):
      Validation: OK
      EPES policy: urn:oid:1.3.6.1.4.1.10015.1000.3.2.1
      SPUri: https://www.sk.ee/repository/bdoc-spec21.pdf
      Signature method: http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
      Signing time: 2013-03-13T08:48:13Z
      Signing cert: MÄNNIK,MARI-LIIS,47101010033
      Produced At: 2013-05-14T23:41:20Z
      OCSP Responder: TEST of SK OCSP RESPONDER 2011
      OCSP Nonce (51): 30 31 30 0D 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20
      10 35 D7 45 F1 42 C1 0C 4D 96 EA 1A 13 C4 34 28 B0 8A 0A 07 47 AA 96 72 0D
      3B 1C C9 02 D0 4B 15

Sample: opening BDOC container, listing its contents and validating signatures
(warnings are displayed as SHA-1 hash function is used in a BDOC file)
> digidoc-tool open weak-sha.bdoc

Input:
  - weak-sha.bdoc                  - input BDOC 2.1 file which contents
                                     are listed and signatures validated
Output:
  Container type: application/vnd.etsi.asic-e+zip
  Documents (1):
    Document (application/octet-stream): test.txt (314 bytes)
  Signatures (1):
    Signature 0 (EPES/time-mark):
      Validation: OK
      EPES policy: urn:oid:1.3.6.1.4.1.10015.1000.3.2.1
      SPUri: https://www.sk.ee/repository/bdoc-spec21.pdf
      Signature method: http://www.w3.org/2000/09/xmldsig#rsa-sha1
```

```
      Signing time: 2012-11-13T11:04:32Z
      Signing cert: MÄNNIK,MARI-LIIS,47101010033
      Produced At: 2012-11-13T11:04:45Z
      OCSP Responder: TEST of SK OCSP RESPONDER 2011
      OCSP Nonce (51): 30 31 30 0D 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20
      10 35 D7 45 F1 42 C1 0C 4D 96 EA 1A 13 C4 34 28 B0 8A 0A 07 47 AA 96 72 0D
      3B 1C C9 02 D0 4B 15
      Warnings: ReferenceDigestWeak, SignatureDigestWeak,

Sample: opening container, extracting its data files
> digidoc-tool open --extractAll demo-container.bdoc

Input:
  - demo-container.bdoc      - input DigiDoc file that is extracted

Output:
  Extracting documents:
    Document(application/octet-stream) extracted to file1.txt (434 bytes)
    Document(application/octet-stream) extracted to file2.pdf (476841 bytes)

Sample: opening container, extracting its data files to a specific directory
> digidoc-tool open --extractAll=demo demo-container.bdoc

Input:
  - demo-container.bdoc      - input DigiDoc file that is extracted

Output:
  Extracting documents:
    Document(application/octet-stream) extracted to demo/file1.txt (434 bytes)
    Document(application/octet-stream) extracted to demo/file2.pdf (476841 bytes)
```

6.3 Adding signatures

Command "sign" enables adding signatures to existing DigiDoc containers. The supported DigiDoc document formats are **DIGIDOC-XML 1.3** and **BDOC 2.1**.

```
> digidoc-tool sign <modified-digidoc-container>
```

Available options:

--city=	Optional City where the signature is created.
--state=	Optional State or province where the signature is created.
--postalCode=	Optional Postal code of the place where the signature is created.
--country=	Optional Country of origin. ISO 3166-type 2-character country codes are used (e.g. EE)
--role=	Optional Signer's role(s). The option can occur multiple times.

--pin=	Optional If PIN is not provided with this parameter value and (the default) PKCS#11 module is used for signing then the utility program asks for the user to insert PIN code to command line during the program's execution time.
--sha(1,224, 256,384,512)	Optional Used for testing purposes. Specifies the hash function that is used for calculating digests. If not specified then SHA-256 is used by default.

Options for specifying module used for accessing the signing token - possible alternatives are PKCS#11 and CryptoAPI/CNG). When signing module is not specified then PKCS#11 module is used by default.

--pkcs11[=]	Optional Signing is done via PKCS#11 module - the default module for singing with smart card in Linux and OS X. When signing via PKCS#11 module then the parameter's value can additionally be used to specify the path and filename of PKCS#11 driver in your file system. For example, "opensc-pkcs11.dll" in Windows environment and "opensc-pkcs11.so" in Linux. If the parameter's value is left unspecified then PKCS#11 driver's location is looked up from configuration file (see also chap. "4.2 Configuration parameters").
--cng	Optional Set the parameter to sign via Microsoft CNG API (in Windows environment). If "--pin" parameter's value is not set then PIN insertion dialog is displayed to the user. Parameter "--cng" may optionally be used along with parameter "--selectFirst".
--selectFirst	Optional Additional parameter that can optionally be used along with parameter "–cng". When the parameter is set then the first certificate in Windows certificate store is chosen for signature creation. If the parameter is not set then certificate selection dialog window is displayed to user.

Note: there is also PKCS#12 signing module support implemented in Libdigidocpp utility program which is used for testing signature creation with ECC keys. More detailed description about usage scenarios is to be determined in the future.

Sample commands for adding signatures:

```
Sample: adding a signature via PKCS#11 driver
> digidoc-tool sign --pkcs11 demo-container.bdoc

Input:
- --pkcs11                  - PKCS#11 module is used for signing
- demo-container.bdoc       - container to be modified

Sample: adding a signature via CNG API
> digidoc-tool sign --cng demo-container.bdoc

Input:
- --cng                     - CNG API is used for signing
- demo-container.bdoc       - container to be modified

Sample: adding a signature via CNG API, no dialog windows are displayed
> digidoc-tool sign --cng --selectFirst --pin=12345 demo-container.bdoc
```

```
Input:
- --cng                     - CNG API is used for signing
- --selectFirst             - the first signing certificate is used for signing
- 12345                     - PIN code (PIN2 in case of Estonian ID cards)
- demo-container.bdoc       - container to be modified
```

6.4 Removing signatures and data files

Signatures and data files can be removed from a DigiDoc container with the command "remove". Note that it is possible to remove data files only from an unsigned container (i.e all signatures must be removed before removing data files). The command is supported with DigiDoc formats **DIGIDOC-XML 1.3** and **BDOC 2.1**.

General format of the command is:

```
> digidoc-tool remove --document=<doc-id> --signature=<sig-id> <modified-digidoc-container>
```

Available options:

--document=	Optional Specifies the sequence number of the data file that is removed from the container. The sequence numbers are counted from zero.
--signature=	Optional Specifies the sequence number of the signature that is removed from the container. The sequence numbers are counted from zero.

Sample commands for removing signatures and data files:

```
Sample: removing signature from container
> digidoc-tool remove --signature=1 demo-container.bdoc

Input:
- 1                         - sequence number of the signature that is removed
- demo-container.bdoc       - container to be modified

Sample: removing data files from container
> digidoc-tool remove --document=0 --document=1 demo-container.bdoc

Input:
- 0                         - sequence number of the data file that is removed
- 1                         - sequence number of the data file that is removed
- demo-container.bdoc       - container to be modified
```

7. National and cross-border support

7.1 Supported Estonian identity tokens

Currently, Libdigidocpp library is tested with the following Estonian ID tokens:

Token	Type	Description	Supported Libdigidocpp functionality
EstEID 3.5, 3.4 and 1.0	Certificate–based PKI smart cards	Different Estonian ID card versions.	All Libdigidocpp functionalities (authentication, signing, verification, encryption/decryption)
Digi-ID (since 2010)	Certificate–based PKI smart card	Estonian Digital ID card for use only in electronic environments	All Libdigidocpp functionalities
Aladdin eToken Pro	Certificate–based PKI USB authenticator	Carrier for ID certificates issued to organizations.	**Note**: Supported and tested using the DigiDoc3 Client application, which is based on the Libdigidocpp library.

7.2 Trusted Estonian Certificate Authorities

AS Sertifitseerimiskeskus (SK, http://sk.ee/en) functions as CA for all the Estonian ID tokens, maintains the electronic infrastructure necessary for issuing and using the ID cards, and develops the associated services and software.

SK issues the certificates and acts as Trusted Service Provider (TSP) for validation of authentication requests and digital signatures. SK maintains the following electronic services for checking certificate validity including:

- **OCSP validation service** (an RFC2560-compliant OCSP server, operating directly off the CA master certificate database and providing validity confirmations to certificates and signatures). There are two ways of getting access the service:
  - having a contract with SK and accessing the service from a specific IP address(es) – as practiced **by companies/services**
  - by having certificate for accessing the service and sending signed requests - as used **by private persons** for giving digital signatures; registering for the service is required and service is limited to 10 signatures per month
- CRL-s (mainly for backward compatibility)
- LDAP directory service (containing all valid certificates)

7.2.1 Supported SK live hierarchy chains

Note: the following certificates are included in the Libdigidocpp distribution package, no additional actions are needed for their usage.

Certificate Common Name (CN)			Valid to	Description
JUUR-SK			26-Aug-2016	SK's 1st root certificate
	ESTEID-SK		13-Jan-2012	for ID cards issued until 2007
		ESTEID-SK OCSP RESPONDER	*24-Mar-2005*	ESTEID-SK OCSP Responder

Certificate Common Name (CN)			Valid to	Description
		ESTEID-SK OCSP RESPONDER 2005	*12-Jan- 2012*	ESTEID-SK OCSP Responder
	ESTEID-SK 2007		26-Aug-2016	for ID cards, Digi-ID and Mobile-IDs issued until 06.2011
		ESTEID-SK 2007 OCSP RESPONDER	*08-Jan-2010*	ESTEID-SK 2007 OCSP Responder
		ESTEID-SK 2007 OCSP RESPONDER 2010	*26-Aug-2016*	ESTEID-SK 2007 OCSP Responder
	EID-SK		08-May-2014	for all other personal certificates issued until 01.2007
		EID-SK 2007 OCSP RESPONDER	*15-May-2007*	EID-SK OCSP Responder
	EID-SK 2007		26-Aug-2016	for Estonian Mobile-IDs issued until 02.2011 and Lithuanian Mobile IDs issued until 06.2011
		EID-SK 2007 OCSP RESPONDER	*17-Apr- 2010*	EID-SK 2007 OCSP Responder
		EID-SK 2007 OCSP RESPONDER 2010	*26-Aug- 2010*	EID-SK 2007 OCSP Responder
	KLASS3-SK		05-May-2012	for organizational certificates issued until 10.2010
		KLASS3-SK OCSP RESPONDER	*05-Apr- 2006*	KLASS3-SK OCSP Responder
		KLASS3-SK OCSP 2006 RESPONDER	*27-Mar-2009*	KLASS3-SK OCSP Responder
		KLASS3-SK OCSP 2009 RESPONDER	*04-May- 2012*	KLASS3-SK OCSP Responder
	KLASS3-SK 2010		26-Aug-2016	for organizational certificates issued from 10.2010
		KLASS3-SK 2010 OCSP RESPONDER	*26-Aug- 2016*	KLASS3-SK 2010 OCSP Responder
EECCRCA			18-Dec- 2030	SK's 2nd root certificate
	ESTEID-SK 2011		18-Mar- 2024	for ID cards, Digi-ID and Mobile-IDs issued from 06.2011
	EID-SK 2011		18-Mar- 2024	for all other personal certificates issued from 06.2011
	KLASS3-SK 2010		18-Mar-2024	for organizational certificates.

Certificate Common Name (CN)			Valid to	Description
	SK OCSP 2011 RESPONDER		*18-Mar- 2024*	common OCSP responder for all certificates issued under EECCRCA

7.2.2 Supported SK test certificate hierarchy chains

Note: in order to use the test certificates with Libdigidocpp, you need to add them separately. The default installation package contains both Estonian and Finnish test certificates (the installation package is accessible from https://installer.id.ee/media/windows/Eesti_ID_kaart_testsertifikaadid.msi). All of the certificates can also be downloaded from http://www.sk.ee/en/repository/certs/.

Note that the test certificates should not be used in live applications as the Libdigidocpp library does not give notifications to the user in case of test signatures.

Certificate Common Name (CN)			Valid to	Description
Test JUUR-SK			27-Aug-2016	SK's 1st test root certificate
	TEST-SK		26-Aug-2016	for all test cards and certificates issued until 04.2011
		Test-SK OCSP RESPONDER 2005	*06-Apr-2012*	TEST-SK OCSP responder
	TEST of KLASS3-SK 2010		21-March-2025	for organizational test certificates
TEST EECCRCA			18-Dec-2030	SK's 2nd test root certificate
	TEST of ESTEID-SK 2011		07-Sep-2023	for test ID cards, Digi-ID and Mobile-ID certificates issued from 04.2011
	TEST of EID-SK 2011		07-Sep-2023	for all other test certificates issued from 04.2011
	Test SK OCSP RESPONDER 2011		*07-Sep-2024*	common OCSP responder for all test certificates issued under TEST-EECCRCA

7.3 Supported Finnish Certificate Authorities

Population Registration Center's Certification Authority Services unit (FINeID, http://fineid.fi/) functions as certificate authority in Finland.

Libdigidocpp supports the following functionality with Finnish certificates and identity tokens:

- signature creation with CAPI/minidriver in Windows environment;
- adding OCSP confirmation to the signature;
    - OCSP confirmation is acquired from AS Sertifitseerimiskeskus proxy OCSP service (http://sk.ee/en/services/validity-confirmation-services/proxy-ocsp/).

The OCSP responder server URL values (http://ocsp.sk.ee/_proxy) have been written directly into the source code.

- signature validation - the CA certificates for signature validation are taken from the Windows certificate store.

Note that the functionality has only been tested in Windows environment.

7.3.1 Supported FINeID live hierarchy chains

Note: in order to use the Finnish certificates with Libdigidocpp, you need to add them separately. The installation package is available from https://installer.id.ee/media/windows/Eesti_ID_kaart_finsertifikaadid.msi

The certificates package contains Finnish root CA certificate (http://fineid.fi/default.aspx?id=596) and certificates which are included in the Finnish national Trust Service List (TSL) (https://www.viestintavirasto.fi/attachments/TSL-Ficora.xml).

Certificate Common Name (CN)		Valid to	Description
VRK Gov. Root CA		18-Dec-2023	1st root certificate
	VRK Gov. CA for Citizen Qualified Certificates	09-Jan-2019	Citizen certificates on identity cards since 2003
	VRK CA for Qualified Certificates	13-Jan-2019	Organization certificates on organization cards since 2003
	VRK CA for Healthcare Professionals Qualified Certificates	17-Dec-2023	Intermediary CA of the certificates for users of the nationwide healthcare information systems

7.3.2 Supported FINeID test certificate hierarchy chains

Note: in order to use the test certificates with Libdigidocpp, you need to install them separately (the installation package which includes both Estonian and Finnish test certificates is accessible from https://installer.id.ee/media/windows/Eesti_ID_kaart_testsertifikaadid.msi). The test certificates are also separately downloadable from http://fineid.fi/default.aspx?id=597.

Note that the test certificates should not be used in live applications as the Libdigidocpp library does not give notifications to the user in case of test signatures.

Certificate Common Name (CN)		Valid to	Description
VRK TEST Root CA		17-Dec-2023	1st test root certificate
	VRK CA for Test Purposes	12-Jan-2019	Test certificates since 2003

8. Interoperability testing

8.1 ASiC Remote Plugtests

The compliance of the BDOC 2.1 containers to ASiC standard [7] was successfully tested in the course of ASiC Remote Plugtests Event (19 November to 07 December 2012). The event aimed to ascertain the correctness and cross usability of ASiC-S and ASiC-E containers created by different participant organizations worldwide. The test cases included generation and cross-verification of ASiC containers with different features (including verification of incorrect files). Each participant chose the appropriate set of tests to be implemented.

The main BDOC 2.1 project coordinator, AS Sertifitseerimiskeskus (SK), participated in the ASiC Remote Plugtests event. The selected test cases were implemented with **Libdigidocpp** (C++) software library of DigiDoc system. Tests that were carried out by SK included functionality that was also applicable for **BDOC 2.1** file format [1]. Features that were tested included generation and cross-verification of **ASiC-E** containers with **XAdES** signatures [5]. The following test cases were covered:

- testing ASiC-E container structure,
- testing ASiC-E container's syntactical conformance,
- testing correctness of XAdES-BES signature in ASiC-E container,
- negative tests of verifying ASiC-E container with invalid XAdES-BES signatures.

The implemented test cases did not cover generation and verification of signatures with time-marks or time-stamps (according to BDOC-TM and BDOC-TS profiles).

Results achieved by SK were as follows:

- test files that were generated by SK were successfully cross-verified by five different participants in 4 out of 6 implemented positive test cases (two of the test files were not verified by other participants).
- SK successfully cross-verified files generated by other three participants in 5 out of 6 implemented positive test cases (one of the files could not be verified because of incompatibility with BDOC 2.1 standard).
- SK successfully passed the negative test case which involved verification of incorrect test file.

Additional information about the ASiC Remote Plugtests event can be found from http://www.etsi.org/plugtests/ASiC/Home.htm.

8.2 DigiDoc framework cross-usability tests

Automated cross-usability tests of digitally signed and encrypted files are periodically carried out between different DigiDoc software libraries [18]:

- Cross-usability tests of digitally signed files in **DIGIDOC-XML 1.3** format (.ddoc files) are carried out between **JDigiDoc** and **CDigiDoc** software libraries.
- Cross-usability of **BDOC 2.1** (.bdoc or .asice) file format is tested between **JDigiDoc** and **Libdigidocpp** libraries.
- Cross-usability of encrypted file format **CDOC 1.0** is carried out between **JDigiDoc** and **CDigiDoc** software libraries.

The interoperability tests are executed through the **command line utility tools of the software libraries** (for example, in case of Libdigidocpp library, the utility program which is described in chapter 6 of the current document).

9. Libdigidocpp implementation notes

The following section describes properties of a BDOC 2.1 file that are not strictly defined in the BDOC 2.1 specification [1] but are used in Libdigidocpp library's implementation (and also in other DigiDoc software libraries) of the file format.

Digital signature related notes:

1. The supported BDOC 2.1 signature profile is a XAdES-EPES profile with time-mark.

   The basic BDOC profile is XAdES-EPES as BDOC 2.1 specification requires that <SignaturePolicyIdentifier> element is present ([1], chap. 5.2).

   It is expected that a time-mark (OCSP confirmation) has been added to the signature as according to BDOC 2.1 specification ([1], chap. 6) a signature is not considered complete or valid without validation data from external services (i.e. a time-mark or time-stamp).

2. Signatures with time-stamps (i.e. BDOC TS profile) are not supported (including archive time-stamps) and will be implemented in the future. Validation data must be added to the signature with a time-mark (according to BDOC 2.1 specification [1], chap. 6.1).

3. One OCSP confirmation (time-mark) is allowed for each signature (due to security reasons and in order to maintain testing efficiency).

4. In case of BDOC signatures with time-mark, the OCSP nonce field's value is calculated as follows:

   - the contents of <SignatureValue> element (i.e. the value without XML tags) is taken and decoded from base64 encoding;
   - digest of the value found in the previous step is calculated by using SHA-256 algorithm.;
   - the digest value found in the previous step and the digest algorithm that was used are transformed as defined by the following ASN.1 structure:

     ```
     TBSDocumentDigest ::= SEQUENCE {
         algorithm AlgorithmIdentifier,
         digest OCTET STRING
     }
     ```

   - the ASN.1 block value produced in the previous step is included in the OCSP request's "nonce" field and must be present in the respective field of the OCSP response.

5. In case of signing with ECC keys (by using ECDSA algorithm), concatenation method is used for creating signature value.

6. In case of BDOC 2.1 documents, SHA-256 hash function is used by default when calculating data file digests and the digest that is signed. In case of Estonian ID cards with certificates issued before 2011, the SHA-224 digest type will be automatically selected and used for calculating signature value's digest (the final digest that is signed), other options are not supported here. Note that other digest in the signature (e.g. data file digests, signer certificate's digest) are still calculated with SHA-256 (the default digest type).

7. When a hash function that is weaker than SHA-256 (or SHA-224 in the special case with pre-2011 ID-cards) has been used then a warning message about weak digest method is produced to the user. It is recommended to regard the error as a validation warning (identically to DigiDoc3. Client application and digidoc-tool.cpp utility program).

8. During signature creation, it is checked that there is only one <ClaimedRole> element in the signature, which contains the signer's role and optionally the signer's resolution. If the <ClaimedRole> element contains both role and resolution then they must be separated with a slash mark, e.g. "role / resolution". Note that when setting the resolution value then role must also be specified.
9. The signature policy document's hash value in <SigPolicyHash> element is checked during validation process (even though it is not mandatory according to BDOC 2.1 specification [1], chap. 5.2). The hash value must correspond to the hash value of the document that is located at https://www.sk.ee/repository/bdoc-spec20.pdf.
10. <Transforms> element is not supported for security purposes and in order to maintain testing efficiency.
11. XML namespace prefixes are used in case of all XML elements (e.g. "asic:", "ds:", "xades:").
12. The data file's MIME type that is used in case of Libdigidocpp's utility program is always "application/octet-stream" for testing purposes.
13. The <ds:Signature> element's Id attribute value is set to "S<seq_no>" during signature creation where sequence numbers are counted from zero. Other Id values that are used in the sub-elements of the <ds:Signature> element contain the signature's Id value as a prefix. During verification, different Id attribute values are also supported but are not tested periodically.

Certificate related notes:

1. Valid signatures (qualified electronic signatures) can be created with a certificate that has "Non-repudiation" value (also referred to as "Content Commitment") in its "Key usage" field. The requirement is based on the following sources:
    - ETSI TS 102 280 (V1.1.1): "X.509 V3 Certificate Profile for Certificates Issued to Natural Persons" [15]; chap. 5.4.3;
    - Profile of certificates issued to private persons by AS Sertifitseerimiskeskus: "Certificates on identity card of Republic of Estonia", version 3.3 [16]; appendix A.3.3;
    - Profile of certificates issued to legal entities by AS Sertifitseerimiskeskus: "Profile of institution certificates and Certificate Revocation Lists", version 1.3 [17]; chap. 3.2.2.
2. Signature can be created with a certificate that doesn't have "Non-repudiation" value in its "Key-Usage" field when specific parameters have been set but validation of such signature will produce a respective error message and the signature is not considered as a qualified electronic signature.
3. During signature validation, it is checked that the validity periods of the signer's certificate and all the certificates in its CA chain include the signature creation time (value of the producedAt field in OCSP response).

Container related notes:

1. BDOC 2.1 files are created with .bdoc file extension. Extensions .asice and .sce are supported and recognized only when reading in an existing BDOC 2.1 file which has the respective extension value.
2. Signatures are stored in META-INF/signatures*.xml files where '*' is a sequence number, counting is started from zero.
3. There can be only one signature in one signatures*.xml file due to BDOC format's legacy issues. Multiple signatures in one signatures*.xml file is not supported in order to maintain testing efficiency. The <ds:Signature> element's Id attribute values

in different signatures*.xml files are generated in the form of "S<seq_no", the sequence numbers are always unique within one BDOC container.

4. It is not allowed to add two data files with the same name to the container as the signed data file must be uniquely identifiable in the container.
5. All data files in the container must be signed. All signatures in the container must sign all of the data files.
6. The META-INF/manifest.xml file's version attribute value is "1.0" (instead of "1.2") as the results of ASiC plug-tests event shows that version 1.0 is used only. The requirement of the OpenDocument version attribute value comes from OpenDocument standard [6] which is referred to in ASiC standard [7].
7. Relative file paths are used, for example "META-INF/signatures*.xml" and "document.txt" instead of "/META-INF/signatures*.xml" and "/document.txt" to ensure better interoperability with third party applications when validating signatures.
8. The ZIP container's comment field contains version number of the library that was used for creating the file. The value can be useful, for example, when trying to determine the origin of an erroneous file.
9. "mimetype" file is not compressed in the BDOC 2.1 file's ZIP container as the results of ASiC plug-tests event shows that this solution is most widely used.
10. When a data file is added to the container then the modification time of the file (as it is registered in the file system) is preserved also in the ZIP container file. There is an exception if the file is added by reading it from an input stream - in that case, the current timestamp value is registered as "last modified" time in the ZIP file.

Appendix 1: Sample Libdigidocpp configuration file

A sample digidocpp.conf file has been provided below. For detailed information about loading configuration settings and the meanings of the configuration parameters, see chapter "4 Configuring Libdigidocpp".

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="schema/conf.xsd">

     <!--Logging settings-->
     <param name="log.level" lock="false">2</param>
    <!--<param name="log.file" lock="false">/tmp/digidocpp.log</param>-->
    <param name="log.file" lock="false">C:\test\CPP\digidocpp.log</param>

     <!--PKCS#11 driver's location, if not using default driver-->
     <param name="pkcs11.driver.path" lock="false">opensc-pkcs11.dll</param>
     <!--<param name="pkcs11.driver.path" lock="false">opensc-pkcs11.so</param>-->

     <!--CA certificates' location, if not using default settings-->
    <param name="cert.store.path" lock="false"></param>

     <!--Location of XML Schema files, if not using the default settings-->
    <param name="xsd.path" lock="false">schema/</param>

    <!--HTTP proxy settings, if needed-->
    <!--<param name="proxy.host" lock="false"></param>-->
    <!--<param name="proxy.port" lock="false"></param>-->
    <!--<param name="proxy.user" lock="false"></param>-->
    <!--<param name="proxy.pass" lock="false"></param>-->

     <!--OCSP request signing options-->
    <param name="pkcs12.cert" lock="false"></param>
    <param name="pkcs12.pass" lock="false"></param>
    <param name="pkcs12.disable" lock="false">false</param>-->

    <!--OCSP responder settings for live OCSP service-->
    <ocsp issuer="ESTEID-SK 2007">http://ocsp.sk.ee</ocsp>
    <ocsp issuer="ESTEID-SK 2011">http://ocsp.sk.ee</ocsp>
    <ocsp issuer="KLASS3-SK 2010">http://ocsp.sk.ee</ocsp>

    <!--OCSP responder settings for test OCSP service-->
    <ocsp issuer="TEST of ESTEID-SK 2011">http://www.openxades.org/cgi-bin/ocsp.cgi</ocsp>
    <ocsp issuer="TEST of KLASS3-SK 2010">http://www.openxades.org/cgi-bin/ocsp.cgi</ocsp>

</configuration>
```

Appendix 2: XML schema modifications

The following section describes modifications that have been made to XML schemas used in Libdigidocpp. The library uses several XML schemas when creating digitally signed documents and validating their structure. The schemas are included in etc/schema/ subdirectory of the Libdigidocpp distribution package, their description has been provided in section "3.5.2 XML Schemas".

Modifications are marked as follows:

- text that has been added is marked as added
- text that has been deleted is marked as ~~deleted~~

1) Schema ts_102918v010201.xsd

- The namespace value has been changed as the initial value in the original schema was incorrect. The value in original schema will be corrected with the next version of the ASiC standard.

```
<xsd:schema targetNamespace="http://uri.etsi.org/29180 2918/v1.2.1#"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        xmlns="http://uri.etsi.org/29180 2918/v1.2.1#"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
        attributeFormDefault="unqualified">
```

- The schema's location has been altered so that the imported schema file is looked up from the local file system.

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
        schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-
        schema.xsd"/>
```

2) Schema xmldsig-core-schema.xsd

- The XMLSchema.dtd reference has been commented out due to implementation issues (otherwise a warning message would be produced).

```
<!--
<!DOCTYPE schema
  PUBLIC "-//W3C//DTD XMLSchema 200102//EN" "http://www.w3.org/2001/XMLSchema.dtd"
 [
   <!ATTLIST schema
     xmlns:ds CDATA #FIXED "http://www.w3.org/2000/09/xmldsig#">
   <!ENTITY dsig 'http://www.w3.org/2000/09/xmldsig#'>
   <!ENTITY % p ''>
   <!ENTITY % s ''>
  ]>
-->
```

- The initial integer data type used in the original schema is converted into long data type when generating C++ source code from the current schema. However, as the

SK issued certificates' serial numbers are too long to fit into long type variable then the data type has been changed to string.

```
<complexType name="X509IssuerSerialType">
  <sequence>
    <element name="X509IssuerName" type="string"/>
    <element name="X509SerialNumber" type="integerstring"/>
  </sequence>
</complexType>
```

3) Schema XAdES.xsd

- The schema's location has been modified so that the file is looked up from the local file system.

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
      schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd"/>xmldsig-core-schema.xsd"/>
```

- The "type" attribute has been added, otherwise a warning message would be produced.

```
<xsd:complexType name="SignaturePolicyIdentifierType">
  <xsd:choice>
    <xsd:element name="SignaturePolicyId" type="SignaturePolicyIdType"/>
    <xsd:element name="SignaturePolicyImplied" type="AnyType"/>
  </xsd:choice>
</xsd:complexType>
```

- The "type" attribute has been added, otherwise a warning message would be produced.

```
<xsd:complexType name="CommitmentTypeIndicationType">
  <xsd:sequence>
    <xsd:element name="CommitmentTypeId" type="ObjectIdentifierType"/>
    <xsd:choice>
      <xsd:element name="ObjectReference" type="xsd:anyURI"
        maxOccurs="unbounded"/>
      <xsd:element name="AllSignedDataObjects" type="AnyType"/>
    </xsd:choice>
    <xsd:element name="CommitmentTypeQualifiers"
      type="CommitmentTypeQualifiersListType" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

- The data type of signer's role has been changed to string due to implementation issues (otherwise the source code generated from the schema would later have to be altered).

```
<xsd:complexType name="ClaimedRolesListType">
```

```
  <xsd:sequence>
    <xsd:element name="ClaimedRole" type="AnyTypexsd:string" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```