Interface Specification

Latvian Post CSP e-Signature Application Software

Prepared for

Latvian Post

Wednesday, 20 December 2006

Version 1.4 Released

Prepared by

Raimonds Rudmanis

Senior Consultant

rairud@microsoft.com

Contributors

Aldis Viļums

Microsoft | Services

Revision and Signoff Sheet

Change Record

Date	Author	Version	Change reference
23.03.2006	Aldis Vilums	.1	Initial draft of the document structure with comments on the contents of each section created for review/discussion
31.03.2006	Raimonds Rudmanis	.2	Initial draft of the document
11.04.2006	Raimonds Rudmanis	.3	Electronic Document Format and Document Validation Service sections
20.04.2006	Raimonds Rudmanis	.4	Updated physical design section
09.06.2006	Raimonds Rudmanis	1.0	Final version
28.06.2006	Raimonds Rudmanis	1.1	Corrections after project Quality Assurance
10.10.2006	Raimonds Rudmanis	1.2	Added EDoc templates and EDoc policies description
31.10.2006	Raimonds Rudmanis	1.3	Updated EDoc format specification
20.12.2006	Raimonds Rudmanis	1.4	Updated EDoc format metadata, templates and signatures description

Reviewers

Name	Version approved	Position	Date
Leonīds Paturskis		Project Manager (Microsoft)	
Aivars Junga		Project Manager (Latvian Post)	

Table of Contents

1 SUMMARY

Interface specification is a design document defining the process, technologies, data and infrastructure required to meet the integration goals between multiple systems.

Interface specification does not define how endpoints integrating are implemented – it defines the requirements that endpoints involved in the integration process should meet and the way in which these endpoints should communicate.

This document describes the specification of the interfaces that can be used by the external parties and customers of the Latvian Post Certification Service Provider (LP CSP) to communicate with public services (authenticated and non-authenticated) exposed by LP CSP information system (IS).

1.1 Goal

The goal of the document is to define the ways in which software developers companies can leverage the services provided by various publicly exposed components of the CSP IS.

1.2 Target Audience

Document is primarily intended for the developers and software architects of companies building software to integrate with Latvian Post CSP IS.

It is also intended for technical and business decision makers that need to understand the approaches and implications of integrating their systems with and using services of Latvian Post CSP IS.

1.3 Document Structure

The document consists of three sections and a summary:

- Summary provides general information on the document including goals, target audience, terms and abbreviations and references used.
- Constraints and Assumptions provides list of constraints and assumptions that were taken into consideration during the design of the integration interfaces. This information provides some context on why some of the design decisions are as they have been defined.
- Overview of Services Provided is a general description of the services exposed to external parties and customers of the Latvian Post CSP. This section allows business and technical decision makers to understand which services they can get from Latvian Post CSP to bring them into the context of applications they are building.
- Supported Scenarios describes:
  - the components of the LP CSP exposed to external parties and customers of the Latvian Post CSP,
  - the components of interest to external parties developing applications.

  The section is considered to provide the overall context and information to non-technical readers.
- Interface Design describes logical interface design of the components that can be interfaced programmatically and how the logical interfaces of the public components defined in logical design are implemented from the point of view of technologies, protocols and security.

1.4 Terms and abbreviations

Terms and abbreviations used in the document are described in the Table 1.

Term or abbreviation	Description
ASN.1	Abstract Syntax Notation number One (notation used for formal description of communication protocols)
CA	Certification authority
CRL	Certificate Revocation List
CSP	Certificate Service Provider
DER	Distinguished Encoding Rules (one of the encodings used in ASN.1)
EDoc	Electronic document format name
IS	Information system
LDAP	Lightweight Directory Access Protocol
LP	Latvian Post
OCSP	Online Certificate Status Protocol
RFC	Request for comments (Internet community document)
TSA	Time stamping Authority
TSP	Time stamping Protocol
XAdES-BES	XAdES Basic Electronic Signature
XAdES-T	XAdES with Time-stamp
XAdES-C	XAdES Complete validation data

Table 1: Terms and abbreviations used in the document

1.5 References

Details on the documents referenced are provided in Table 2.

#	Referenced document details
1.	Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C., X.509 Internet Public Key Infrastructure: Online Certificate Status Protocol - OCSP. RFC 2560, June 1999
2.	C. Adams, P. Cain, D. Pinkas, R. Zuccherato, Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP), Network Working Group, RFC 3161, August 2001.
3.	Eastlake 3rd D., Reagle J., Solo D., (Extensible Markup Language) XML Signature Syntax and Processing. (XML-DSIG), RFC 3275, March 2002
4.	XML Advanced Electronic Signatures (XAdES), ETSI TS 101 903, v 1.3.2, March 2006
5.	XML Schema Part 2: Datatypes. W3C Recommendation, XML Schema 2, 02 May 2001
6.	Open Packaging Conventions, Specification and Reference Guide, version 0.85, 2006
7.	SOAP Message Transmission Optimization Mechanism, W3C Recommendation 25 January 2005
8.	Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), OASIS Standard Specification, 1 February 2006
9.	ITU-T Rec. X.501, "The Directory: Models", 1993
10.	ITU-T Rec. X.511, "The Directory: Abstract Service Definition", 1993

Table 2: Documents referenced

The Open Packaging Conventions specification, version 0.85 is published on Microsoft web site.

2 CONSTRAINTS AND ASSUMPTIONS

List of constraints and assumptions on which the interface design is based and technologies where chosen are defined in Table 3.

#	Description
1.	Networking services are available for using LP CSP online services
2.	
3.	

Table 3: Constraints and assumptions taken into account during the interface design

3 OVERVIEW OF SERVICES PROVIDED

3.1 Solution perspective

Since year 2001 Latvian government had been developed e-government strategy, which provides a roadmap and priorities for development of e-services in different areas. LR Electronic Documents Law regulates the way of electronic documents usage in the territory of Latvia and specifically in areas regulated by central and local governments. This law is in force since January 1st, 2004.

On the year 2003 government of Latvia had set priority to make e-services portfolio available for Latvian residents in 4 years. One of the prerequisites of making e-services available is to have Certification Service Provider in place. Latvijas Pasts management had made a decision to expand its services portfolio by offering certification services to Latvian residents and government organizations in the territory of Latvia. This decision had been extended by partnering with Lattelekom who agreed to become a hosting organization of LP CSP IS by providing its Data Center facilities and computing resources.

3.2 Services provided

LP CSP IS solution provides certification services for Latvia government and residents which enables the usage of electronic documents in different areas. VAS Latvijas Pasts (LP) offers certification services for use with various business applications that need user authentication or digital signature functionality.

Latvian Post CSP IS public services are displayed in Figure 1.

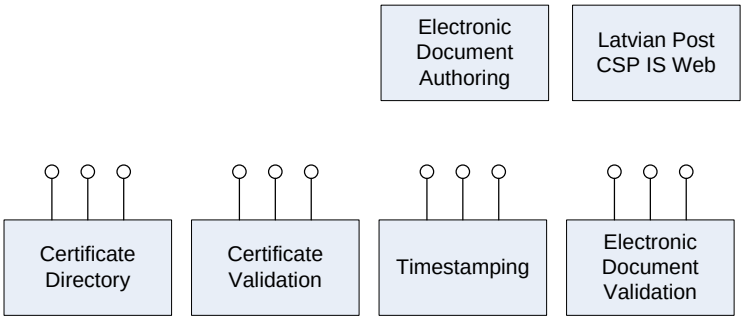

Figure 1: Latvian Post CSP IS public services

LP CSP IS solution exposes the following public services:

- certificate directory services;
- time stamping services (for authenticated users only);
- certificate validation services;
- electronic document authoring services (as standalone application);
- digital signatures and electronic documents validation services.

Additionally LP CSP maintains a Website that provides access to the following information and services:

- access to the Certificate Validation Services (for verification of certificate validity);
- access to the Directory Services (for searching and downloading certificates);
- electronic document validation services;
- access to documents directly related to the operation of the LP CSP CA and RA;

- downloadable software for certificate usage.

For customers of the Latvian Post CSP, Website provides additional functionality of:

- viewing system log information relative the customer;
- changing customer details (e.g., contact information) online.

In this document the Website will not be described in details because it is not exposing interfaces for external systems - only for physical persons.

3.3 Latvian Post CSP IS user entities

User entities of the LP CSP are the ones that use LP CSP solution. Those include:

LP CSP Customers are residents of the Latvia who have applied for and received smart card and/or certificates from LP CSP.

These users can use LP CSP system to author and validate electronic documents (as defined by the Electronic Document Law of the Republic of Latvia). Documents can be signed on-line or offline using certificate and private key information issued to customer by LP CSP. If trusted timestamp is required, access to LP CSP is required during signature creation process. Document validation can be performed both offline (using CRL information published in the LDAP directory) and on-line (using certificate status information from OCSP responder).

LP CSP customers can use their certificates and private keys to authenticate into:

- LP CSP IS Web site and receive services provided there;
- any other external system that trusts LP CSP and accepts LP CSP issued certificates as login credentials.

They can also use LP CSP IS services available on-line to get information about certificates of other LP CSP customers and to validate certificates of other LP CSP customers.

Relying parties are external users and systems that use services of the LP CSP IS or certificates and CRLs issued by the LP CSP IS to:

- validate documents signed by the LP CSP Customers;
- search and validate certificates of LP CSP Customers in LDAP certificates directory;
- authenticate user accessing system using LP CSP issued authentication certificate when relying party is external system.

Because parties cannot be authenticated (they are not registered by LP CSP IS), relying party services are publicly available.

4 SUPPORTED SCENARIOS

4.1 Certificates Description

The LP CSP CA offers two types of certificates - one for authentication and one qualified for signature creation and verification, together on a smart card. There are also two associated private keys, protected by two separate PIN codes, on the card. Certificates issued by LP CSP CA are suitable for authentication purposes and to support digital signing.

Certificate applicability conforms to the qualified certificate requirements stated by the LR Electronic Documents Law. Certificates are issued and signed by LP CSP CA and consist of the attributes in compliance with the LR Electronic Documents Law.

Qualified certificate contains the following mandatory attributes:

- User notice, that the certificate is qualified;
- LP CSP organization name, registration number and country;
- Subject first name, last name or pseudonym;
- Subject personal code, assigned to a person by Latvian Residents Register (Iedzīvotāju reģistrs);
- Certificate validity period;
- Unique certificate ID;
- Subject RSA public key.

All certificates are signed by LP CSP CA.

Certificates issued by LP CSP must be used according to the X.509 v3 key usage field as set by LP CSP. Certificates may contain additional attributes depending on certificate usage policy.

4.1.1 Identification and Authentication of Subscriber

4.1.1.1 Type of names

All Certificates contain X.509 Distinguished Names in the **Issuer** field. The following attributes are applied:

- countryName = LV
- organizationName = VAS Latvijas Pasts, Vien. reģ. Nr. 40003052790
- organizationalUnitName = Uzticams sertifikācijas pakalpojumu sniedzējs
- commonName = VAS Latvijas Pasts Saknes sertifikācijas institūcija | VAS Latvijas Pasts Politikas sertifikācijas institūcija | VAS Latvijas Pasts Drošo sertifikātu dienests

All Certificates contain X.509 Distinguished Names in the **Subject** field. The following attributes are applied:

Personal certificates (contain the name of a natural person)

- countryName = <country of Subject (the ISO 3166 code)>
- givenName = <Given name (as written in the document of identification)>
- surName = <surname (as written in the document of identification)>
- serialNumber = <unique ID Number (personal code or other system generated unique ID1)>
- optional: organizationName = <Subject's Organization>

- optional: organizationalUnitName = <Subject's Organizational Unit>]

4.1.1.2 Pseudonyms of Subscribers

The Subject may choose a pseudonym during the certificate application process. If the Subject wants to use a pseudonym for qualified certificate instead of a name, it is to be indicated that is a pseudonym (names other than a Subject's true personal or organizational name by adding a prefix "PN: ".

In case of using pseudonyms, the chosen pseudonym will be put in the commonName of Subject DN. Surname and givenName will not been used then.

The use of pseudonyms for Subject authentication certificates is not permitted.

4.2 Certificate Directory

Certificate Directory Service is a service of the LP CSP which provides online access via LDAP to the following information:

- Certificate and CRL of the LP Root CA;
- Certificate and CRL of the LP Policy CA;
- Certificate and CRL of the LP CSP CA;
- Certificate of the LP CSP TSA;
- Qualified certificates for signature for which the Subject has given permission to be made publicly available;
- Authentication certificates for which the Subject has given permission to be made publicly available;
- Certificate of the OCSP-Responder(s).

Only active certificates will be available via LDAP in certificate directory. Suspended, revoked or expired certificates will not be available through this interface.

Certificate Directory service provides the required functionality of publishing certificates. It contains CA and TSA certificates and active certificates issued to LP CSP clients for which the client has given permission to be made publicly available. Certificate Directory contains also Certificate Revocation Lists (CRL) for Latvian Post CAs and TSA. LP CSP services are responsible for maintaining information in the directory up-to-date. Certificate Directory service is public and therefore is available to callers using LDAPv3 over SSL.

Certificate Directory services can be used by third party applications for searching and collecting certificates and for downloading CRLs. It can be used also for certificate validation process by applications which are not supporting real-time certificate validation protocols like OCSP (see 4.3).

Searches are the most common certificate directory activity. The LDAP interface lets you use a wide variety of search criteria and result-retrieval methods to find directory information. Among other things, this provides everyone a chance of finding the e-mail address of LP CSP clients, if they have chosen to publish certificates and to include their e-mail addresses in them.

When doing searching in Certificate Directory the caller can specify:

- distinguished name of the entry at which to start the search;
- the scope of the search;
- the search filter.

Typical certificate directory search process is displayed in Figure 5.

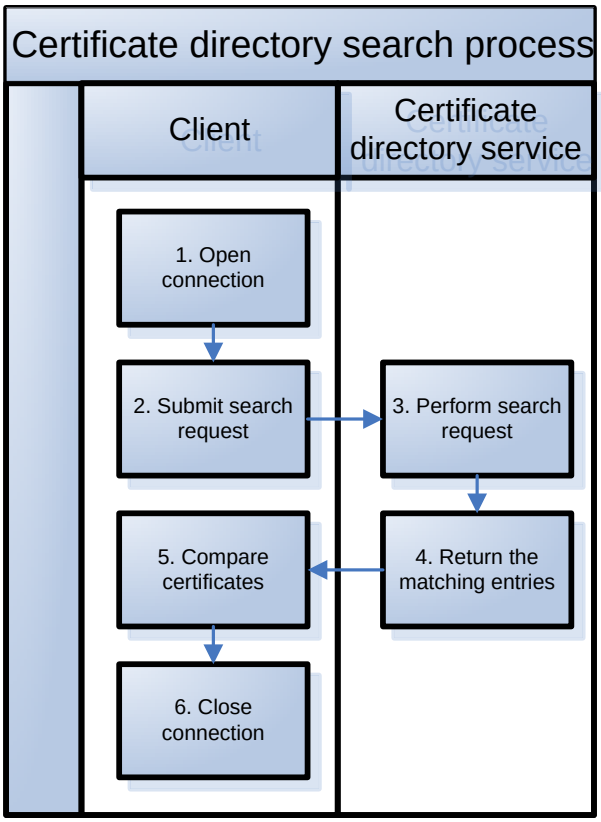

Figure 2: Certificate directory search process

Certificate directory search process consists of the following steps:

1) **Open connection**. The client opens a TCP connection to an LDAP server and submits a bind request. The bind operation includes the name of the directory entry;
2) **Submit search request**. The client submits a search request to the server;
3) **Perform search request**. The server performs the search request;
4) **Return the matching entries**. The server returns the matching entries to the client;
5) **Compare certificates**. The client can compare the certificate received with the one which requires validation;
6) **Close connection**. The client submits an unbind request to the server and closes the connection.

4.3 Certificate Validation

Certificate validation ensures that the certificate's information is authentic, that the certificate is used only for its intended purposes, and that the certificate is trusted.

LP CSP provides three ways of checking certificate validity:

1) LDAP directory contains all active certificates for which the client has given permission to be made publicly available. The directory is updated close to real time with a delay for publishing – if a certificate is activated, it becomes available in the directory, and, when it is revoked or suspended, it is removed from there. Restrictions are in effect as to the maximum number of responses returned to one LDAP query to protect against server overload.
   However this is not the suggested method, because there will be valid certificates not

published in the directory and this will cause problems for those customers that do not publish their certificates;

2) CRLs contain the list of suspended and revoked certificates. CRLs are standard but outdated method, because they can grow very fast in size and it is not convenient to use them from third party applications. CRLs are provided mainly for backwards compatibility and standards compliance. Delta CRLs are also provided by LP CSP.
This is suggested method for offline validation of the certificate status;

3) Alternative is to use online validation of certificates by using Online Certificate Status Protocol (OCSP). OCSP responses reflect the actual real-time certificate status. It operates directly with master LP CSP CA certificate database and does not use CRLs. It is also most convenient method of verifying certificate validity;
This is suggested method of validating certificate status.

Conceptual certificate **offline** validation schema is displayed in Figure 3.

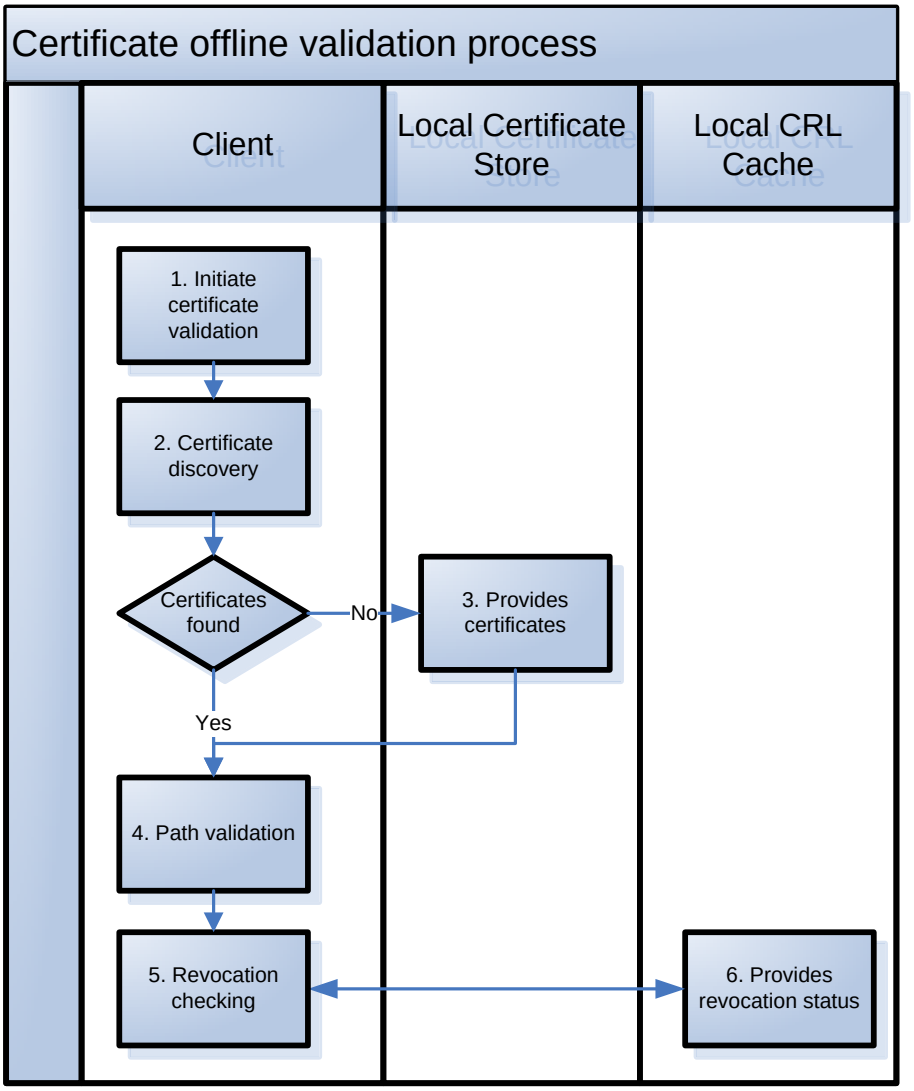

Figure 3: Conceptual certificate offline validation schema

Conceptual certificate validation process consists of the following steps:

1) **Initiate certificate validation**. The client initiates certificate validation process. When a certificate is presented to an application, the application must use the certificate chaining engine of operating system to determine the certificate's validity;

2) **Certificate discovery**. To build certificate chains, the certificate chaining engine must collect the issuing CA certificate and all CA certificates up to the root CA certificate. Normally full chain of certificates will be included already as part of electronic signature;
3) **Collect certificates from Local Certificate Store**. In case the CA certificates are not available in signature they can be collected from Local Certificate Store, if they have been added to Trusted Root Certification Authorities folder of the Local Certificate Store. This can be done by operating system automatically;
4) **Path validation**. Each certificate in the certificate chain must be validated until a self-signed root certificate is reached. Validation tests can include verifying signatures of certificates, checking for specific application or certificate policy and determining whether certificates are included in particular certificate stores;
5) **Revocation checking**. In case of offline validation the application checks revocation status for each certificate against the CRL;
6) **Access local CRLs**. CRLs are retrieved from local operating system cache. The cached CRLs are used to prevent excess network traffic. Because CRLs are cached for some time, this method doesn't provide real-time certificate status.

Conceptual certificate **online** validation schema is displayed in Figure 4.

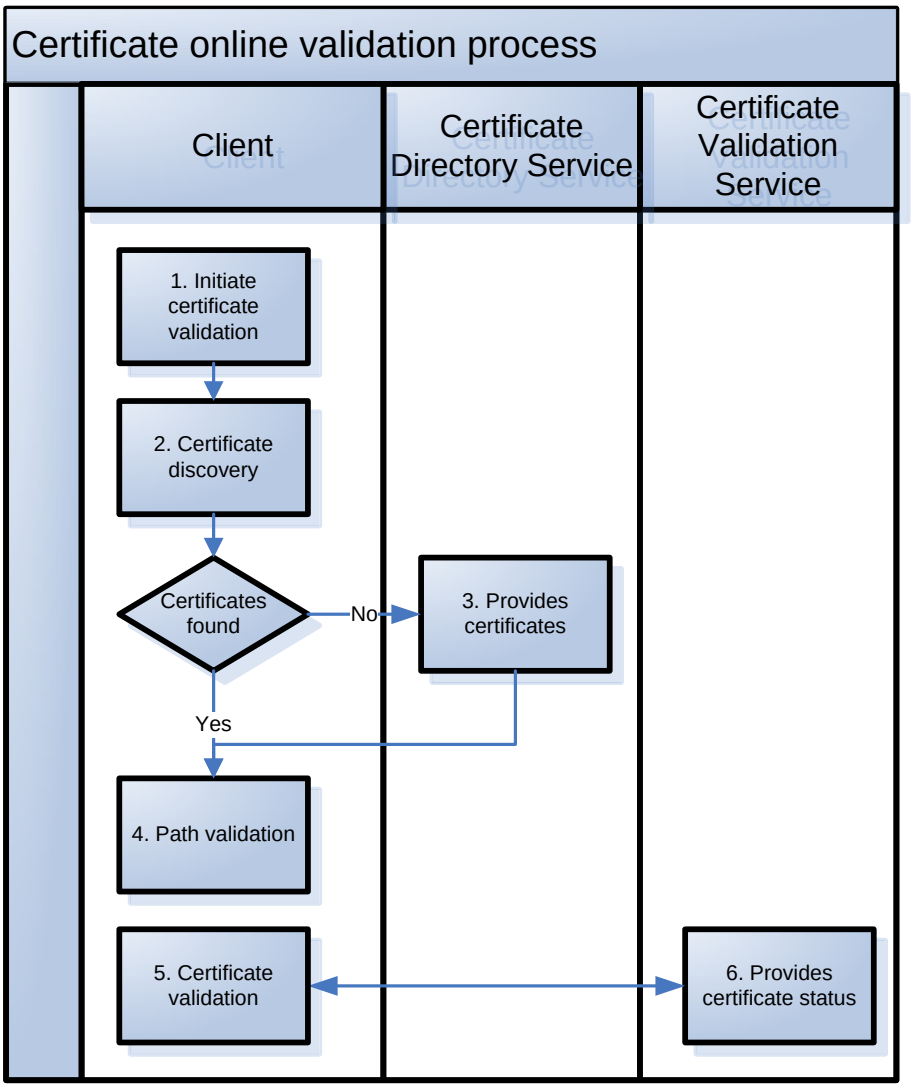

Figure 4: Conceptual online certificate validation schema

Conceptual certificate validation process consists of the following steps:

1) **Initiate certificate validation**. The client initiates certificate validation process. When a certificate is presented to an application, the application must use the certificate chaining engine of operating system to determine the certificate's validity;
2) **Certificate discovery**. To build certificate chains, the certificate chaining engine must collect the issuing CA certificate and all CA certificates up to the root CA certificate. This can be done by third party application itself. Usually it is done by operating system;
3) **Collect certificates from Certificate Directory Service**. CA certificates can be collected from Certificate Directory Service or operating system cache;
4) **Path validation**. Each certificate in the certificate chain must be validated until a self-signed root certificate is reached. Validation tests can include verifying signatures of certificates, checking for specific application or certificate policy and determining whether certificates are included in particular certificate stores;
5) **Certificate validation**. Each certificate in the certificate chain must be validated until a self-signed root certificate is reached. In online scenario the certificate status is requested by means of OCSP requests;
6) **Collect certificate status**. OCSP responder provides a real-time certificate status.

Certificate Validation Service provides online access to following information:

- Certificate Status of Root CA;
- Status of certificates issued by Root CA;
- Status of certificates issued by all Policy CAs;
- Status of certificates issued by all Issuing CAs.

All certificates are at any time verifiable by OCSP Validation Service. The OCSP Responder of LP CSP is based upon Online Certificate Status Protocol (OCSP) according to RFC 2560.

4.4 Time stamping

Important documents in real life (purchase orders, insurance contracts or partnership agreements) are always first dated, and then signed. Doing so, the signer acknowledges both the document is authentic and its date is valid.

Electronic document time stamping sticks to the same philosophy: a date is first appended to the document and then digitally signed. The process protects the document against both content and date forgery. A time stamp is simply a data unit that proves that certain data existed at a certain moment.

TSA component is responsible for providing the time stamping functionality to the clients of the LP CSP. It provides time stamping services for authenticated clients of the Post CSP IS. Time stamping services are based upon TSP (time stamping protocol) defined by the RFC 3161 standard, which are extended to support authentication of the timestamp requestor. LP CSP time stamping services can be used by third party applications for time stamping the electronic documents. It is used also by standalone electronic document signing application provided by Latvian Post.

Time stamping process is displayed in Figure 5. Basically, two different entities are involved in time stamping process:

- authenticated user:
  - sends over a digitally signed time stamp request asking for one of his documents (to be precise – document or signature hash) to be time stamped;

- verifies the received timestamp and TSA signature upon response;

- a server, named time stamp authority (TSA), is responsible for time stamping. It is trusted to maintain an accurate clock and sends back a time stamp response signed by the timestamp authority.

Time stamp request

The structure of a time stamp request is very simple. It contains the hash of the document to time stamp, a protocol version number and some more optional parameters, specified in physical design section. Note the hash of the document is sent across to TSA, not the full document. As a matter of fact, for confidentiality and/or efficiency reasons, client might not wish to send over his data in clear text.

LP CSP IS extends the basic request by requiring it to be signed by the caller. This is needed for authentication purposes.

Time stamp response

The time stamp response structure is much more complicated: the document hash originally received by TSA, current date, unique response ID and optionally TSA's certificate. The response is signed using TSA private key,

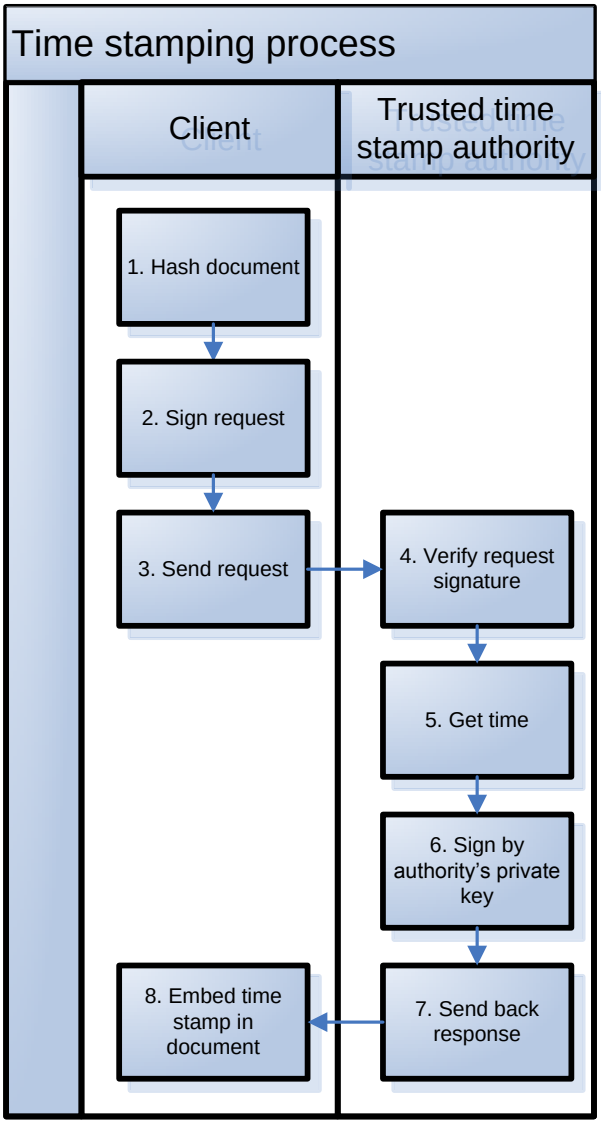

Figure 5: Time stamping process

Time stamping process consists of the following steps:

1) **Hash document**. Hash is calculated for the document to be time stamped;
2) **Sign request**. The time stamp request has to be signed by caller for authentication purposes;
3) **Send request**. Time stamping request is sent to the trusted time stamp authority, using standard HTTPS protocol;
4) **Verify request signature**. Signature and attached certificate of time stamp request is verified by the time stamping authority. Signer's certificate is verified whether it is still valid during the time stamping process;
5) **Get time**. Trusted time stamp authority is responsible for getting accurate clock;
6) **Sign by authority's private key**. Times stamp resulting structure is encapsulated in a cryptographic message which is signed by the TSA with its private key. TSA certificate is embedded in the response as well;
7) **Send back response**. The time stamp is received back to caller by standard HTTPS protocol;
8) **Embed time stamp in document**. Resulting time stamp is usually embedded in the document to be time stamped. Then the whole document is signed by user's private key.

LP CSP provides time stamping services only for authenticated clients of the Post CSP IS.

In the Post CSP IS solution authentication for the time stamping service is implemented by wrapping the TSA request into PKCS#7 package signed using customer's non-repudiation certificate (the same used for signing the document to be time stamped).

4.5 Electronic Document Authoring

LP CSP provides standalone application for document authoring and signing (signer application) as well as electronic document format specification. The application is distributed for free.

4.5.1 Signer application

The application for document authoring and signing provides the following functionality:

- creating electronic document, corresponding to format specified by LP CSP;
- time stamping the electronic document;
- digitally signing the electronic document;
- verifying signature and time stamp.

Signer application is supported on Microsoft Windows 2000 and Microsoft Windows XP operating systems. It is standalone rich client application.

The typical electronic document **authoring and signing process** consists of the following steps:

1) The user launch application;
2) Choose EDoc template;
3) Choose one or multiple documents from file system;
4) Specify metadata for electronic document;
5) Choose the signature type (with or without time stamp);
6) Application requests the smartcard to be inserted in the reader;

7) User enters PIN code;
8) Application then forms the document according to specification, requests and adds time stamp to the document un digitally signs it;
9) User can save this document into file system or send it by e-mail to other party.

The typical electronic document **time stamp and signature verification process** consists of the following steps:

1) The user launch application;
2) Choose and open electronic document;
3) If the document format is correct, user can examine the original documents, metadata, time stamps and signatures, otherwise user will receive error message;
4) User can verify the time stamp and signature;
5) Application will check the status of signatures by means of OCSP responder or CRL if user is not online;
6) User can extract and save original documents into file system.

4.5.2 Electronic document format

Electronic document (EDoc) is hierarchical container, which contains:

- EDoc metadata;
- Original documents (files);
- Zero or more signatures.

Electronic document schematically is displayed in Figure 6.

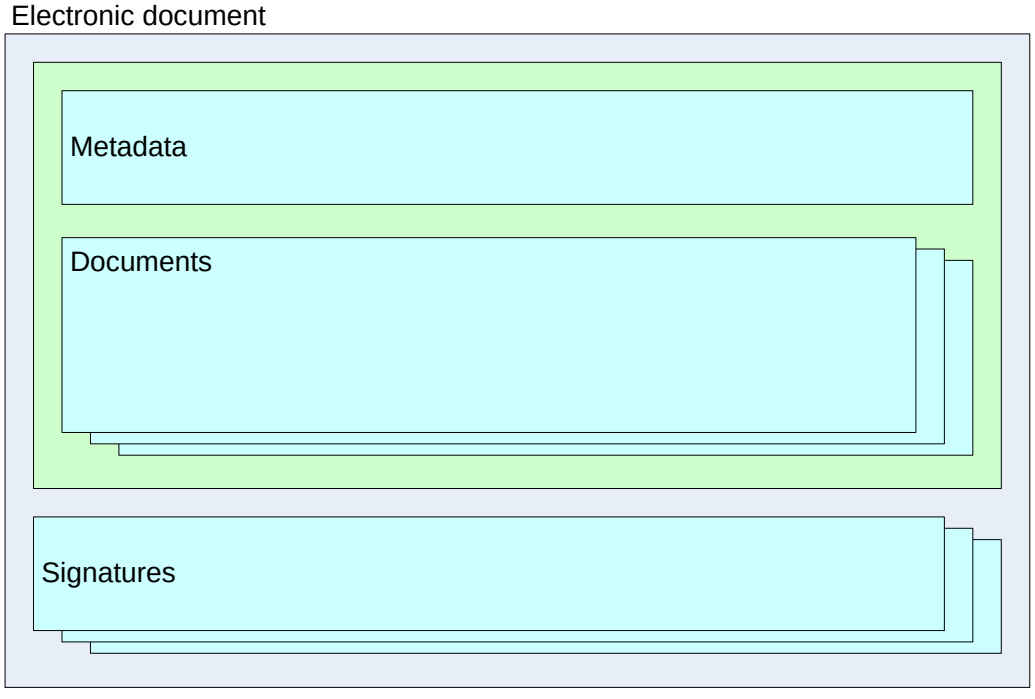

Figure 6: Electronic document format

EDoc uses a file container that conforms to the Open Packaging Conventions. The Open Packaging Conventions, which describes the method for packaging information in a file format, describing metadata, parts, relationships and application of digital signatures. The Open Packaging

Conventions use open technologies such as XML and the ZIP archive file format and is documented as part of initial draft of the Ecma standard for the Open XML Formats.

Signatures embedded in the document are based on the international standards XML-DSIG and ETSI TS 101 903. XML Advanced Electronic Signatures (XAdES) [ETSI TS 101 903] defines a format that enables structurally store signed data, signature and security attributes associated with digital signature (e.g. validity confirmations).

Electronic signature format of EDoc can be based on XAdES-BES, XAdES-T or XAdES-C profiles. Signature profile can be selected for each signature by user during document signing. The list of allowed signature profiles for specific business document is defined by EDoc template used to create the document.

XAdES-BES profile defines the basic electronic signature without a time stamp.

The main distinguished feature of XAdES-T profile is a presence of qualified time stamp. An electronic signature with timestamp adds the time-stamp to the electronic signature which is the initial step to the long-term validity of the signature. Input to the time-stamping process is the digital signature value. The XML equivalent to this structure is XAdES-T.

XML Advanced Electronic Signature with Complete validation data references (XAdES-C) additionally includes full chain of certificates used for signature as well as full certificates revocation information. This type of signature provides the highest security level as it includes all information required to validate signature even offline without certificate services involvement.

In order for a digital signature to be considered technically valid (according to the ISO definition of digital signature), the signature must be created while the signer's certificate is valid. Time-stamp can prove that fact.

EDoc format provides the following features:

- Signature can be given to one or many documents at the same time;
- Signature can be used to sign the metadata;
- One or many signatures can be attached per container;
- Original document can be structured XML document or any binary file (Word, Excel, PDF etc).

EDoc format in future can be an interface between:

- users creating documents in signer application and sending them to users having third party application;
- users creating documents of the same format in third party application and sending them to users having application distributed by LP.

An **electronic signature** has two types of components: some components must be present (mandatory) in the electronic signature (as required by ETSI TS 101 733), while others are optional. Some of those components (when present) must be protected by a digital signature. The following Table 4 provides a list of signature components, which are mandatory in EDoc signatures. All components must be signed, i.e. protected by a digital signature.

Component Name	Component Description
Signing time	This is the time of signature as claimed by the signer.
Certificate Identifier	The signer’s Certificate could be contained within certificates field of the electronic signature structure.
Reference to signature policy	Signature policy is a set of rules the signer and verifier have to follow in order to get the same result when evaluating an electronic signature.

Table 4: Overview of digital signature components

4.5.3 Multiple Signatures

In general electronic signatures can be sequential or parallel (cosigning).

Co-signature

In case of parallel signature document (shown in Figure 7) each signature signs document (documents) and signed metadata sections. A co-signature is used when all parties agree on the content being signed. The validity of other signatures is not checked or confirmed. In the electronic signature structure each co-signature appears as an additional structure. All signature information structures are at the same level.

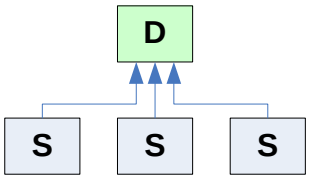

Figure 7: Parallel signature document

Counter-signature

In case of sequential signature document (shown in Figure 8) first signature signs document (documents) and signed metadata sections, other signatures sign previous signature in the list. By using a counter-signature the signer confirms the content being signed and also confirms the earlier signature.

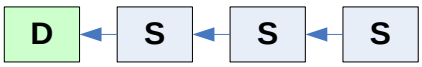

Figure 8: Sequential signature document

EDoc solution in current version supports only counter signatures.

4.5.4 Electronic document validation process

EDoc validation consists of multiple individual checks. Each check can return the following status:

- If check is successful, it gets "**PASSED**" (or green) status;
- If check failed, it gets "**FAILED**" (or red) status. In the case of failed check additional information about the reason of failure is returned to the application or user;
- "**NOT CHECKED**" (or grey) status indicates that the check has not failed but there is insufficient information to determine if the check is valid. For example; OCSP service is not available. In this case EDoc can be validated again at some later time when additional validation information becomes available. Also, in the case of incomplete validation, additional information is returned to the application or user, thus allowing the application or user to decide what to do with partially correct EDoc.

EDoc validation consists of the following checks:

		Document			Signature		
Item#	**Check description**	**Passed**	**Failed**	**Not checked**	**Passed**	**Failed**	**Not checked**
1.0 Verify EDoc							
1.1	Verify EDoc against EDoc general schema. Only success or fail possible.	Valid	Invalid	-	-	-	-

1.2	Verify EDoc document metadata against template. If template with the same ID missed in local cache than it should be downloaded from URL; if failed than final status is Not checked.	Valid	Invalid	Valid with warnings	-	-	-
2.0 Verify signatures							
2.1	Check signature structure matches declared signature type	Valid	Invalid		Valid	Invalid	
2.2	Verify EDoc signature metadata against template. If template with the same ID missed in local cache than it should be downloaded from URL; if failed than final status is Not checked;	Valid	Valid with warnings	Valid with warnings	Valid	Invalid	Valid with warnings
2.3	Check if signed hash matches the hash of the signed part	Valid	Invalid		Valid	Invalid	
3.1 XADES-BES and XADES-T signature checks							
3.1.1	Build certificate chain, check if Root is trusted	Valid	Invalid	-	Valid	Invalid	
3.1.2	Check status of all certificates in the chain except root using OCSP. Suppose that OCSP-R certificate is checked here as well using CRL.	Valid	Invalid	Check 3.1.3	Valid	Invalid	Check 3.1.3
3.1.3	If 3.1.2. returns Not Checked check all the certificate chain (except root) using CRLs (Offline CRL check)	Valid	Invalid	Invalid	Valid with warnings	Invalid	Invalid
3.2 XADES-T signature checks (in addition to XADES-BES)							
3.2.1	Check Timestamp integrity (validate signature hash against timestamped hash)	Valid	Invalid	-	Valid	Invalid	
3.2.2	Check TSA certificate using CRL (Offline CRL check)	Valid	Invalid	Invalid	Valid	Invalid	Invalid
3.3 XADES-C signature checks							

3.3.1	Build certificate chain using certificate information in XADES-C signature (ensures that chain can be built, status of certs is not checked)	Valid	Invalid	Invalid	Valid	Invalid	Invalid
3.3.2	Check the Root of the chain is trusted Root	Valid	Invalid	Invalid	Valid	Invalid	Invalid
3.3.3	Check OCSP responses included in XADES-C contain valid status for all certificates in the chain	Valid	Invalid	Invalid	Valid	Invalid	Invalid
3.3.4	Build OCSP-R certificate chain (take OCSP certificate from OCSP Response), ensure it is correct and ends up with the same root as signer chain.	Valid	Invalid	Invalid	Valid	Invalid	Invalid
3.3.5	Check that no certificate in OCSP-R chain is revoked (do not check root because root should be trusted) by looking at CRLs stored inside XADES-C.	Valid	Invalid	Invalid	Valid	Invalid	Invalid
3.3.6	Check Timestamp integrity (validate signature hash against timestamped hash)	Valid	Invalid	Invalid	Valid	Invalid	Invalid
3.3.6	Build TSA certificate chain (using TSA certificate in Timestamp), ensure it can be built	Valid	Invalid	Invalid	Valid	Invalid	Invalid
3.3.7	Check that TSA chain ends up in trusted root.	Valid	Invalid	Invalid	Valid	Invalid	Invalid
3.3.8	Check that no certificate in TSA chain is revoked (do not check root because root should be trusted) by looking at CRLs stored inside XADES-C.	Valid	Invalid	Invalid	Valid	Invalid	Invalid
3.3.9	Validate that time for OCSP resonses used to validate signer chain is withing grace period from time in TSA timestamp	Valid	Invalid	Invalid	Valid	Invalid	Invalid

3.3.10	Validate that CRLs used to validate TSA certificate chain were valid at the time in the TSA timestamp	Valid	Invalid	Invalid	Valid	Invalid	Invalid
3.3.11	Validate that CRLs used to validate OCSP-R certificate chaing were valid at the time in the TSA timestamp	Valid	Invalid	Invalid	Valid	Invalid	Invalid
Summary		EDoc is Valid if all are Valid	EDoc is Invalid if any of those are Invalid	According to results above	Signature is Valid if all are Valid	Signature is Invalid if any of those are Invalid	According to results above

Table 5: EDoc validation checks

It is possible to perform full validation of EDoc with all attached signatures or to validate each particular signature separately in which case only appropriate list of checks will be performed (see Signatures subsections in table 5).

As a result of EDoc validation output status of the EDoc or signature validation process can be:

- "**VALID**" (or green) status. A Valid response indicates that the EDoc has passed verification and it complies with the EDoc validation policy;
- "**VALID with WARNINGS**" (or yellow) status. Typically it means that some minor checks have failed or returned "NOT CHECKED" status. It allows the application or user to decide what to do with partially correct EDoc or signature. The exact meaning of this status is defined by EDoc validation policy;
- "**INVALID**" (or red) status. An Invalid response indicates that the EDoc has failed verification according to EDoc validation policy.

4.5.5 Electronic document validation policies

EDoc validation policies are required to establish technical consistency when validating EDoc. When a comprehensive EDoc policy used by the verifier is either explicitly indicated by the signer or implied by the data being signed, then a consistent result can be obtained when validating the EDoc and attached signatures. When the EDoc policy being used by the verifier is neither indicated by the signer nor can be derived from other data then verifiers, including arbitrators, may obtain different results when validating an EDoc. Therefore, comprehensive EDoc policies that ensure consistency of EDoc validation are recommended from both the signers and verifiers point of view.

The EDoc validation policy defines a number of rules that have to be followed by both the signer when producing the EDoc and by the verifier when verifying such an EDoc. A given legal/contractual context may recognize a particular EDoc validation policy as meeting its requirements. Currently EDoc library supports only one built-in policy described in chapter 4.5.4.

4.6 Server-side Document Validation

LP CSP provides two mechanisms of server-side document validation:

- user is able to upload the electronic document to LP CSP IS Website and validate the document there;
- for external systems server-side document validation interface is provided.

LP CSP IS Website document validation service can be used in the following scenarios:

- the user receives digitally signed document and he doesn't have the offline application available for document validation;
- document can be validated from internet café or kiosk;
- there is no offline application available for the operating system the user is using.

For external systems, e.g. e-services providers, LP CSP provides XML Web Services based interface, for document validation. It allows customers submitting documents for validation, reviewing the validation status and getting the validation result. This service is asynchronous. It will receive the document from caller and issue the caller unique ticket identifying the particular request. Then caller can periodically poll the service for response. When ready the service will return the status of document submitted.

LP CSP provides server-side document validation XML Web Services only for authenticated users of the Post CSP IS. For caller authentication HTTPS session based on X.509 certificate will be used.

5 ELECTRONIC DOCUMENT FORMAT

Electronic document (EDoc) uses a file container that conforms to the Open Packaging Conventions, which describes the method for packaging information in a file format, describing metadata, parts, relationships and application of digital signatures. EDoc implements a subset of Open Packaging Conventions features.

5.1 Open Packaging Conventions overview

5.1.1 The Package

At the highest level, each EDoc document is stored in a package, meaning a ZIP archive containing a bunch of parts and items. A package is a logical entity that holds a collection of parts. The purpose of the package is to aggregate all of the pieces of a EDoc (or other type of content) into a single object. The parts are composed, processed, and persisted according to a set of rules. Parts have relationships to each other, as well as to the package itself. Parts have content types and are unambiguously addressed using the well-defined naming guidelines.

Because the EDoc format will continue to evolve, packages are designed to accommodate extensions and support compatibility goals.

The Open Packaging Conventions specification, version 0.85 is published on Microsoft web site.

Package sample is displayed in Figure 9.

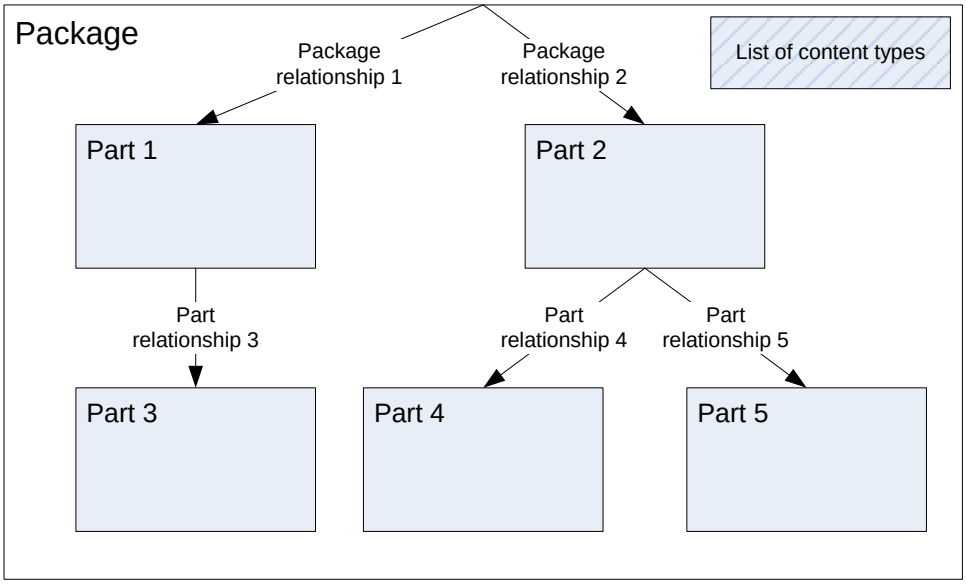

Figure 9: Package sample

Parts are the chunks of content that make up the document, while items are descriptive metadata that defines how the parts are assembled and rendered.

Items can be divided into:

- relationship items (defining how parts are related);
- content-type items (defining the content type of each part).

Relationship items can be further divided into:

- package relationships (relating a part to the package);
- part relationships (relating one part to another).

Package mapping to a ZIP Archive

A ZIP archive is a ZIP file as defined in the ZIP file format specification but excluding all elements of that specification related to encryption or decryption. A ZIP archive contains ZIP items. ZIP items become files when the archive is unzipped. When users unzip a ZIP-based package, they see a set of files and folders that reflects the parts in the package and their hierarchical naming structure.

5.1.2 Parts

Each document has a main part that is the "outermost" part, and all other parts (if any) are contained or referenced within this main part. Parts may be any type of binary content.

A part is a stream of bytes with a MIME content type. A stream is a linearly ordered sequence of bytes. Parts are analogous to a file in a file system.

Each part has a name. Part names refer to parts within a package, typically as part of a URI reference. Like URIs, part names are represented by a logical hierarchy that consists of segments. Part names are case-insensitive.

Every part has a content type, which identifies the type of content that is stored in the part. Content types define a media type, a subtype, and an optional set of parameters. Content types for package parts MUST fit the definition and syntax for media types as specified in RFC 2616, Section 3.7.

All XML content of the parts defined in this specification MUST be encoded using UTF-8.

5.1.3 Relationship Items

The package introduces a higher-level mechanism to describe references from parts to other internal or external resources: relationships. Relationships represent the type of connection between a source part and a target resource. They make the connection directly discoverable without looking at the part contents, so they are independent of content-specific schemas and quick to resolve.

This relationship is stored in the ".rels" file in the "_rels" folder. The "_rels" folder is a fundamental concept of the packaging convention. Relationships are always defined in a "_rels" folder that is directly subordinate to the folder of the source of the relationship item.

Consumers MUST NOT attempt to locate a part via a well-known part name. Consumers MUST use these package relationships to locate parts.

5.1.4 Content-Type Items

Content types are metadata describing the file type of each part in the package. Typical content types may include simple embedded objects such as "text/plain" or "image/jpeg" as well as more expansive types such as "application/xml". Note that even the relationships items have a content type (application/vnd.ms-package.relationships+xml). Content types tell the consumer of an EDoc package how to interpret or render each of the parts.

Content types are all stored together in a single item named "[Content_Types].xml" in the root folder of the package. The types in this package apply to all of the parts throughout all levels of the package's internal hierarchy.

A default content type is typically associated with a file extension, such as xml or jpg. Override content types may also specify that a specific part is a specific content type, regardless of its file extension.

5.2 EDoc format description

5.2.1 EDoc package

EDoc package is ZIP file with extension ".edoc". All EDoc specific parts are located in folder "/EDoc". Additionally EDoc will use core properties part defined in Open Packaging Conventions. Core properties part is located in "/docProps/Core.xml" file. Core properties include a variety of metadata such as the package creator's name, the package creation time, and search keywords.

Sample EDoc package, consisting of two Microsoft Word documents and two attached signatures is shown in Figure 10. Signed package parts are shown with striped background. Signed relationships are made bold.

After the first signature is added, all documents with corresponding relationships are signed and no new document can be added to the EDoc package. Only core properties can be changed and additional signatures can be added without breaking EDoc integrity.

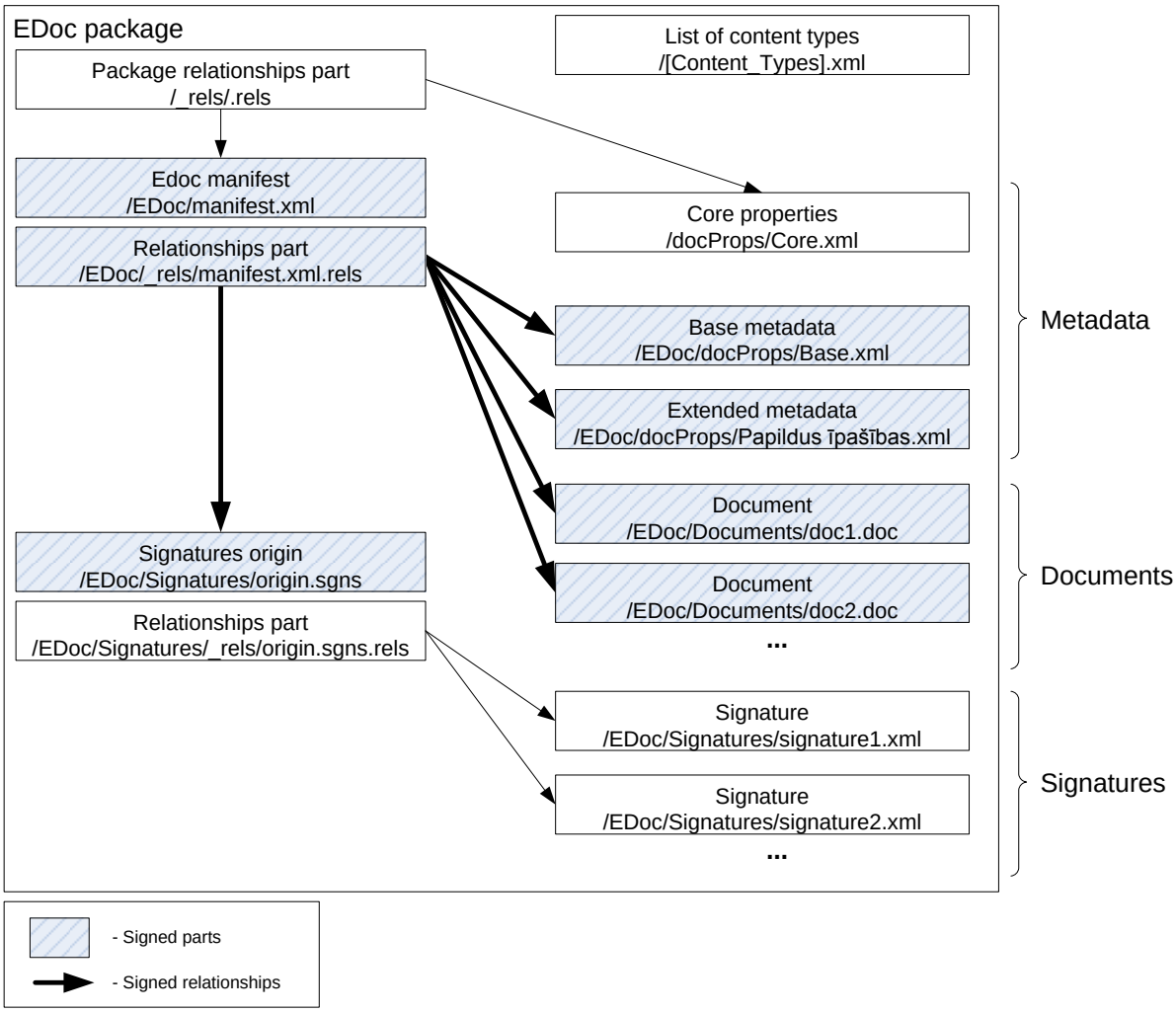

Figure 10: EDoc package sample

EDoc package consists of the following parts and items:

Part name	Description
/[Content_Types].xml	Contains EDoc content types. It is not signed by EDoc signature to

	allow adding new files and therefore new types to the package.
/_rels/.rels	EDoc package relationships to the parts. It contains two relationships – to /docProps/core.xml and to /EDoc/manifest.xml. Package relationships are not signed to allow extending EDoc format with new root relationships.
/docProps/Core.xml	EDoc core properties defined in Open Packaging Conventions. Core properties are not signed.
/EDoc/manifest.xml	EDoc manifest describes document type, EDoc format and version.
/EDoc/_rels/manifest.xml.rels	Relationships from EDoc manifest to each individual file containg metadata, documents (files) and signatures origin. All EDoc documents are located in /EDoc/Documents folder. manifest.xml has relationships to the all metadata and documents contained in EDoc.
/EDoc/Signatures/origin.sgns	Starting point for navigating through the signatures in a package. All EDoc signatures are located in /EDoc/Signatures folder. Signatures are named sequentially: signature1.xml, signature2.xml … signatureN.xml. File origin.sgns has relationships to the all signature parts. This file must be empty according to OpenXML standard.
/EDoc/Signatures/_rels/origin.sgns.rels	Relationships from signatures origin to each individual signatures.

Table 6: EDoc package parts

5.2.2 Content types

Content types are stored in "/[Content_Types].xml" file.

The Content Types stream contains XML with a top-level <Types> element, and one or more <Default> and <Override> child elements. <Default> elements define default mappings from the extensions of part names (that is, file extensions) to content types. (File extensions often correspond to content type.) <Override> elements are used to specify content types on parts that are not covered by, or are not consistent with, the default mappings. Package producers MAY use pre-defined <Default> elements to reduce the number of <Override> elements on a part, but are not required to do so.

The Content Types stream maps content types and package part names.

Name	Description
<Types>	The root element of the Content Types stream.
<Default>	Defines default mappings from the extensions of part names to content types.
<Override>	Specifies content types on parts that are not covered by, or are not consistent with, the default mappings.

Table 7: Content Types stream elements

Name	Description	Required/Optional
Extension	A part name extension. A <Default> element matches any part whose name ends with a period followed by the value of this attribute.	REQUIRED
ContentType	A content type as defined in RFC 2616. Indicates the content type of any matching parts (unless overridden).	REQUIRED

Table 8: <Default> attributes

Name	Description	Required/Optional
PartName	A part name. An <Override> element matches the part whose name is equal to the value of this attribute.	REQUIRED
ContentType	A content type as defined in RFC 2616. Indicates the content type of the matching	REQUIRED

	part.

Table 9: <Override> attributes

Sample EDoc content types will contain the following elements:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
<Default Extension="rels"
     ContentType="application/vnd.openxmlformats-package.relationships+xml"/>
<Default Extension="xml" ContentType="application/xml"/>
<Default Extension="doc" ContentType="application/word"/>
<Default Extension="sgns"
     ContentType="application/vnd.microsoft-edoc.signature-origin"/>
<Override PartName="/docProps/core.xml"
     ContentType="application/vnd.openxmlformats-package.core-properties+xml"/>
<Override PartName="/EDoc/manifest.xml"
     ContentType="application/vnd.microsoft-edoc.manifest+xml"/>
<Override PartName="/EDoc/Signatures/signature1.xml"
     ContentType="application/vnd.microsoft-edoc.signature+xml"/>
<Override PartName="/EDoc/Signatures/signature2.xml"
     ContentType="application/vnd.microsoft-edoc.signature+xml"/>
</Types>
```

Content types schema is included in APPENDIX A.

5.2.3 Relationships

Relationships are represented using <Relationship> elements nested in a single <Relationships> element.

Name	Description
<Relationships>	The root element of the Relationships part.
<Relationship>	Represents a single relationship. Every relationship MUST have an Id attribute, the value of which must be unique within the Relationships part.

Table 10: Relationship elements

Name	Description	Required/Optional
Target	A URI reference pointing to a target resource. The URI reference may be a URI or a relative reference.	REQUIRED
Id	MUST be a valid XML identifier. The Id type is xsd:ID and must conform to the naming restrictions for xsd:ID. (See the W3C Recommendation “XML Schema Part 2: Datatypes.”)	REQUIRED
Type	A URI that uniquely defines the role of the relationship.	REQUIRED

Table 11: <Relationship> element attributes

Sample EDoc package relationships file (/_rels/.rels) will contain the following elements:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship
     Id="rId1"
     Type="http://schemas.openxmlformats.org/package/2006/relationships/metadata/core-
properties"
     Target="/docProps/core.xml"/>
<Relationship
     Id="rId2"
     Type="http://schemas.microsoft.com/edoc/2006/manifest"
     Target="/EDoc/manifest.xml"/>
</Relationships>
```

Sample EDoc manifest relationships file (/EDoc/_rels/manifest.xml.rels) will contain the following elements:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship
     Id="rId1"
     Type="http://schemas.microsoft.com/edoc/2006/signature/origin"
     Target="/EDoc/Signatures/origin.sgns"/>
<Relationship
     Id="rId2
     Type="http://schemas.microsoft.com/edoc/2006/metadata/signed"
     Target="/EDoc/docProps/Base.xml"/>
<Relationship
     Id="rId3
     Type="http://schemas.microsoft.com/edoc/2006/metadata/signed"
     Target="/EDoc/docProps/Papildus īpašības.xml"/>
<Relationship
     Id="rId4
     Type="http://schemas.microsoft.com/edoc/2006/document"
     Target="/EDoc/Documents/doc1.doc"/>
<Relationship
     Id="rId5
     Type="http://schemas.microsoft.com/edoc/2006/document"
     Target="/EDoc/Documents/doc2.doc"/>
</Relationships>
```

Sample EDoc signatures origin relationships file (/EDoc/Signatures/_rels/origin.sgns.rels) will contain the following elements:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship
     Id="rId1"
     Type="http://schemas.microsoft.com/edoc/2006/signature"
     Target="signature1.xml"/>
<Relationship
     Id="rId2"
     Type="http://schemas.microsoft.com/edoc/2006/signature"
     Target="signature2.xml"/>
</Relationships>
```

Relationships schema is included in APPENDIX B.

5.2.4 EDoc manifest

EDoc manifest contains EDoc format name, format version and document type.

Sample EDoc manifest file (/EDoc/manifest.xml) will contain the following elements:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Manifest xmlns="http://schemas.microsoft.com/edoc/2006/manifest">
  <DocumentType>URN:MICROSOFT:LV:EDOCTEMPLATE:EXTENDED</DocumentType>
  <TemplateLocation>http://server/folder/Template.edot</TemplateLocation>
  <Format>EDOC-XML</Format>
  <Version>1.01</Version>
<Manifest>
```

The root element of EDoc manifest is <Manifest>. It contains the following elements:

- DocumentType – URN, which describes the document template, by default it is URN:MICROSOFT:LV:EDOCTEMPLATE:EXTENDED, which points to default template, see APPENDIX H;
- TemplateLocation – template location URL address;
- Format – EDoc file format name. Current format is "EDOC-XML";
- Version – EDoc file format version. Current version is "1.01".

EDoc manifest schema is included in APPENDIX D.

5.2.5 Templates

EDoc template describes document properties and signatures. With template mechanism organizations can define their own custom business document and signature templates with additional metadata, which are saved in extended properties section of EDoc. Template is an XML document that complies with the schema EdocTemplate.xsd (see APPENDIX E).

Template contains the following information:

- Template unique identifier;
- General information about the template (organization, description, version);
- Name of the metadata section that template defines (There are multiple metadata sections, each described with its own template – see the next section);
- URL address, where the original template is located;
- List of the properties. For each property, template specifies its name, XML namespace, data type, allowed values, whether or not the property can be changed by the user and whether its mandatory;
- Optional list of signature types. Each signature type has name and its own set of signature properties.

Properties of EDoc metadata sections are defined in the appropriate template. Metadata sections are validated against the corresponding template, when a document is opened and during EDoc validation process.

For extended properties section EDoc signer is downloading the templates from URL address specified in EDoc manifest and storing them in local cache. Templates are stored in organizations, which is issuing the templates, web location. Alternatively templates can be stored on organizations local file server for internal use. Templates are never saved as part of EDoc document.

Document type, which uniquely identifies template and template location are specified in EDoc manifest <DocumentType> and <TemplateLocation> elements. <DocumentType> is URN specifying relationship between document and corresponding template. Each template has unique template identifier, which is referenced by <DocumentType> in EDoc. Additionally each template has “sectionName” attribute which sets a name of the metadata section created.
Templates for core properties and base properties are hardcoded in EDoc library and they are not being downloaded from URL address.

Templates are cached locally in “templates\cache” subfolder by signer application. To find a template, signer application:

- is looking first in local cache by template unique identifier (URN);
- if not found, application is showing user the warning, that template is not found and asking, whether it should be downloaded from the URL specified in EDoc;
- if user chooses not to download, application is using default hard coded template;
- if user chooses to download, application is downloading template and caching it locally.

If the template can not be downloaded from the URL specified, the default EDoc template is applied to EDoc extended properties, see APPENDIX H. Default EDoc template does not contain any properties.

5.2.6 Metadata

EDoc metadata is divided into multiple sections, each containing multiple properties. Each metadata section of the document is described by a template. Structure of core properties and base properties sections is hardcoded in EDoc solution and can not be changed. However organizations can define their own extended properties sections of EDoc metadata by using templates mechanism.

EDoc format does not limit a number of these sections and templates; however, EDoc library always creates documents with three metadata sections:

- **Core properties** are defined by OpenXml standard. They are described by the core template. That template is never saved with the document. It is hardcoded in the resources of EDoc library. Template definition can be found in APPENDIX F;
- **Base properties** are properties that every EDoc has. They are described by the base template. That template is also hardcoded in the resources of EDoc library. Template definition can be found in APPENDIX G;
- **Extended properties** are specific for each organization that uses EDocs. User selects template for extended properties when he creates the document. Allowed signature types are also defined in extended template. Extended properties structure is defined by using template mechanism.

5.2.6.1 Core properties

Core properties enable users to get and set well-known and common sets of properties within packages. Core properties are stored in /docProps/core.xml file. Some of these properties are automatically set by the application and cannot be changed by the user. Core properties used by EDoc applications are a subset of OpenXml core properties. They are described by Core template (see APPENDIX F).

Property	Namespace	Description
Created	Dublin Core	Date of creation of the resource.
Creator	Dublin Core	An entity primarily responsible for making the content of the resource.
Language	Dublin Core	A language of the intellectual content of the resource.
Last_Modified_By	Open Packaging Conventions	The user who performed the last modification. The identification is environment-specific and can consist, for example, of a name, email address, or employee ID. It is recommended that this value be as concise as possible.
Modified	Dublin Core	Date on which the resource was changed.
Revision	Open Packaging Conventions	The revision number. This value indicates the number of saves or revisions. The application is responsible for updating this value after each revision.
Title	Dublin Core	The name given to the resource.

Version	Open Packaging Conventions	The version number. This value is set by the user or by the application.

Table 12: Core properties

5.2.6.2 Base properties

Base properties are a feature of EDoc format (unlike core properties which are defined in OpenXml specification). They are defined in base template.xml (see appendix G) and stored in /Edoc/docProps/Base.xml. At the moment there is only one base property:

Property	Namespace	Description
eDoc_Format	edoc	Format version. E.g. 1.0.0

Table 13: Base properties

5.2.6.3 Extended properties

Extended properties are defined by using the template mechanism described in 5.2.5.

Extended properties are stored in section Edoc/docProps/<sectionName>.xml where sectionName is taken from extended template's sectionName attribute, typically it is "Papildus īpašības".

Content of extended properties depends on the extended template used when document is created. Identifier of template is stored in EDoc manifest <DocumentType> element.

5.2.7 Signatures

Signatures embedded in the document are based on the international standards XML-DSIG and ETSI TS 101 903. XML Advanced Electronic Signatures (XAdES) [ETSI TS 101 903] defines a format that enables structurally store signed data, signature and security attributes associated with digital signature (e.g. validity confirmations).

Electronic signature format is based on XAdES-BES, XAdES-T or XAdES-C profile, depending of signature type chosen by the signer: basic, timestamped or verified. XAdES-BES, XAdES-T or XAdES-C profiles are used in EDoc according to XAdES specification.

Starting point for navigating through the signatures in a package is the Digital Signature Origin part (/EDoc/Signatures/origin.sgns file). This file has zero or more relationships to signatures files. Each signature is stored in separate file in folder /Edoc/Signatures.

Only one Digital Signature Origin part is allowed in a package and it MUST be targeted from the EDoc manifest using the well-defined relationship type with value: "http://schemas.microsoft.com/edoc/2006/signature/origin".

Relationships to the Digital Signature XML Signature parts are defined in the Relationships part. The contents of a Digital Signature Origin part itself SHOULD be empty.

5.2.7.1 Electronic signature with complete validation data references (XAdES-C)

If the signer chooses verified signature type, a XAdES-C profile signature is created.

In this case CompleteCertificateRefs and CompleteRevocationRefs elements are also incorporated in the signature, containing references to all certificates and revocation values that have been used in the validation of the electronic signature.

CompleteCertificateRefs element contains a sequence of references to the full set of CA certificates that have been used to validate the electronic signature up to (but not including) the signing certificate.

CompleteRevocationRefs element contains a full set of references to the revocation data that have been used in the validation of the signer and CA certificates.

According to XAdES standard validation data can be stored outside the signature to prevent redundant storage and to reduce the size of the signatures, however in EDoc format validation data itself is also added to the signature for archiving purposes. It is necessary because certificate authority currently is not providing archiving services.

All the certificates from the certification path, starting with those from the signer and ending up with those of the certificate from one trusted root are captured and stored in CertificateValues element.

Additionally CRL and OCSP response is stored in RevocationValues element.

Therefore EDoc signature contains all mandatory elements from XAdES-C profile plus CertificateValues and RevocationValues elements.

In order to safeguard against the possibility of a CA key in the certificate chain ever being compromised electronic signatures incorporating time-stamps on validation data references are also needed. It is currently not implemented in EDoc format because timestamp requests are not free of charge.

5.2.7.2 Signature parts

Signature parts are based on XAdES schema, which is part of XML Advanced Electronic Signatures (XAdES) specification.

EDoc signature part for basic signature consists of <SignedInfo>, <SignatureValue>, <KeyInfo> and multiple <Object> elements.

For each document section being signed, a XAdES <Object> is added to the signature XML. This object contains relative path to EDoc part and its digest (hash) value, see APPENDIX I. Additionally there is one object containing XAdES signature metadata and one object containing EDoc signature metadata, see APPENDIX J. SignedInfo element contains references to all XAdES objects being signed together with digest (hash) value, see Figure 11 as a simplified illustration of XAdES references. Each <Reference> element has an URI attribute. A value of the URI attribute is “#<object-id>”.

Figure 11: EDoc signature references

XAdES signature is then created over this digest value, not the actual document content. This means that signature verification involves not only XAdES verification (to verify that the signature and digests were not tampered with) but also verification of digests against the actual document data (to verify that the document itself has not been replaced).

<SignedInfo>, <SignatureValue>, <KeyInfo> and XAdES <Object> elements are used according to the XAdES standard. <SignedInfo> is XML-DSIG block that contains the info to be signed, <SignatureValue> contains actual signature, <KeyInfo> contains the certificate used to give the signature and its RSA public key.

XAdES object Id attributes

Each signature element, which is being referenced by other elements, has Id attribute of type xsd:ID. Id attribute is used to make a reference to this element.

Each root <Signature> element has Id value the same as corresponding signature part name - EDoc\Signatures\signatureN.xml, where N is the sequential number of signature starting from 1.

<SignatureValue> element has Id with value SignatureValue_EDoc\Signatures\signatureN.xml, where N is the sequential number of signature starting from 1.

Objects, which are referencing EDoc parts has Id with the following format: “hash_” + <relative path to EDoc part>, e.g. “hash_EDoc\docProps\Base.xml”.

Object, containing EDoc signature metadata, has the following Id: “signature_properties”.

Object, containing XAdES signature metadata, has the following Id: “XadesObject_EDoc\Signature\signatureN.xml” , where N is the sequential number of signature starting from 1. XAdES object <SignedProperties> subelement has the following Id:

SignedProperties_EDoc\Signature\signatureN.xml, where N is the sequential number of signature starting from 1.

5.2.7.3 Signing EDoc parts

The following EDoc parts and relationships are always signed as shown in Figure 10.

Description	Part name
EDoc manifest	/EDoc/manifest.xml
Relationships from EDoc manifest	/EDoc/_rels/manifest.xml.rels
Digital signatures origin	/EDoc/Signatures/origin.sgns
Base metadata	/EDoc/docProps/Base.xml
Extended metadata	/EDoc/docProps/<sectionName>.xml, where sectionName is defined as part of the template used when EDoc is created. The section name for default template is: “Papildus īpašības”.
Documents	/EDoc/Documents/*.*

Table 14: EDoc package signed parts

The first signature is signing all EDoc parts listed in the Table 14. Every next signature added to the EDoc is signing all EDoc parts listed in the Table 14 plus additionally all previous signature parts. This approach corresponds to counter signatures schema described in 4.5.3. All signatures are chained. It means that no signatures can be detached from EDoc without invalidating all consecutive signatures. Only the last signature can be detached from EDoc without invalidating all other signatures.

EDoc manifest with all outgoing relationships is always signed. Core metadata are not signed and therefore can be changed even after document signing without invalidating the signatures. Digital signatures origin is signed.

Base metadata is signed. Extended metadata parts which are referenced by relationships with the type “http://schemas.microsoft.com/edoc/2006/metadata/signed” will be signed. Extended metadata parts referenced by relationships with the type “http://schemas.microsoft.com/edoc/2006/metadata/unsigned” will not be signed.

All documents referenced by relationships with type “http://schemas.microsoft.com/edoc/2006/document” will be signed.

Document templates are never saved with the document and therefore are not signed.

5.3 Differences between EDoc format versions 1.0 and 1.01

The document format version 1.01 fixes some minor errors from 1.0:

- In version 1.0 EDoc signatures origin relationships file: /EDoc/Signatures/_rels/origin.sgns.rels was missing. There was no relationship from EDoc manifest to signatures origin and from signatures origin to individual signatures. Instead there were relationships directly from manifest to all individual signatures;
- In version 1.0 the following parts were not signed:
  - manifest - /EDoc/manifest.xml;
  - manifest relationships - /EDoc/_rels/manifest.xml.rels;
  - signatures origin - /EDoc/Signatures/origin.sgns;
- In version 1.0 folder name for signatures was “Signature”, in version 1.01 it is “Signatures”;

- In version 1.0 the relationship type for signed properties was:
  "http://schemas.microsoft.com/edoc/2006/metadata/unsigned"

  In version 1.01 the relationship type for signed properties is:
  "http://schemas.microsoft.com/edoc/2006/metadata/signed"
- In version 1.0 origin.sgns file was always present and not empty – 1 byte in size;
- In version 1.0 extended properties part name was not URL-encoded;
- In version 1.0 The content of [Content_Types].xml file was incorrect. The following item was wrong:
  <Default Extension="rels" ContentType="application/vnd.microsoft-edoc.signature-origin" />

  The following items were missing:

  <Default Extension="sgns" ContentType="application/vnd.microsoft-edoc.signature-origin"/>

  <Override PartName="/docProps/core.xml" ContentType="application/vnd.openxmlformats-package.core-properties+xml"/>

  <Override PartName="/Edoc/manifest.xml" ContentType="application/vnd.microsoft-edoc.manifest+xml"/>
- In version 1.0 manifest.xml had xmlns="" for each element. TemplateLocation was specified also for default hardcoded template, which was not available at the URL specified.

6 INTERFACE DESIGN

6.1 LDAP

Certificate Directory service is available to callers using LDAPv3 over SSL.

These services of LP CSP are available online at:

- ldap://e-me.lv and
- http://www.e-me.lv

6.1.1 LDAP structure for Certificate Services and Application

The following diagram represents LDAP catalogue structure which will be publicly available for Certificate Services and Applications.

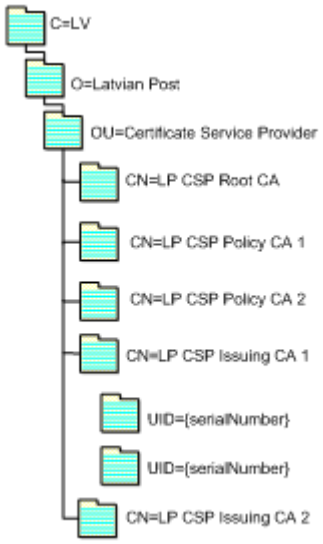

Figure 12: LDAP catalogue structure

The following table list Object classes and theirs purpose which will be used within LDAP directory.

Path	Object class	Purpose
C=LV	Country	Store information for whole country in Latvian Post LDAP directory
O=Latvian Post	Organization	Store information for whole Latvian Post organization
OU=Certificate Service Provider	Organization Unit	Store information for whole Certificate Service Provider
CN=LP CSP Root CA	Certification Authority	Store CA certificate information and distribution list published by Root CA
CN=LP CSP Policy CA 1	Certification Authority	Store CA certificate information and distribution list published by Policy CA 1
CN=LP CSP Policy CA 2	Certification Authority	Store CA certificate information and distribution list published by non

		Policy CA 2
CN=LP CSP Issuing CA 1	Certification Authority	Store CA certificate information, distribution lists (base and delta) published by Issuing CA 1. This object will also collect information of all subject signature certificates (public key) issued by this CA stored in inetOrgPerson objects
CN=LP CSP Issuing CA 2	Certification Authority	Store CA certificate information, distribution lists (base and delta) published by Issuing CA 2. This object will also collect information of all subject signature certificates (public key) issued by this CA stored in inetOrgPerson objects
UID=xxxxx	inetOtgPerson	Store information of end user certificate. Value of UID attribute will equal value of certificate serial number

Table 15: Description of used LDAP object classes

Subject signature certificates should be searched starting from “CN=LP CSP Issuing CA 1” or “CN=LP CSP Issuing CA 2” directories.

6.1.2 LDAP Search Operation

Only search operation against LDAP directory is allowed. The Search operation allows a client to request that a search be performed on its behalf by a server. This can be used to read attributes from a single entry.

Only one of two following combinations for LDAP queries is allowed:

- Subject name + Subject surname + Subject personal code;
- Subject name + Subject surname + Certificate serial number.

All other queries will be blocked from public access.

The logical interface of LDAP Search operation is defined in Table 16.

Method	Interface	Description
Search	In: ▪ LDAP search request (binary blob) Out: ▪ LDAP search response (binary blob)	Actions: ▪ Upon receipt of a search request, a server will perform the necessary search of the Directory Information Tree (DIT) ▪ The server will return to the client a response in a separate LDAP message

Table 16: LDAP Search operation interface definition

LDAP search request contains the following attributes:

Attribute	Data Type	Mandatory?	Comments

baseObject	LDAP Distinguished Name	☑	An LDAP Distinguished Name that is the base object entry relative to which the search is to be performed. Detailed description of LDAP Distinguished Name is defined in RFC 2253
scope	baseObject (0), singleLevel (1), wholeSubtree (2)	☑	An indicator of the scope of the search to be performed. The semantics of the possible values of this field are identical to the semantics of the scope field in the X.511 [10] Search Operation
derefAliases	neverDerefAliases (0), derefInSearching (1), derefFindingBaseObj (2)	☑	An indicator as to how alias objects (as defined in X.501 [9]) are to be handled in searching
sizeLimit	INTEGER	☑	A sizelimit that restricts the maximum number of entries to be returned as a result of the search. A value of 0 in this field indicates that no client-requested sizelimit restrictions are in effect for the search. Servers may enforce a maximum number of entries to return
timeLimit	INTEGER	☑	A timelimit that restricts the maximum time (in seconds) allowed for a search. A value of 0 in this field indicates that no client-requested timelimit restrictions are in effect for the search
typesOnly	BOOLEAN	☑	An indicator as to whether search results will contain both attribute types and values, or just attribute types. Setting this field to TRUE causes only attribute types (no values) to be returned. Setting this field to FALSE causes both attribute types and values to be returned
filter	Filter	☑	A filter that defines the conditions that must be fulfilled in order for the search to match a given entry. Detailed description of Filter is defined in RFC 2251 and RFC 2254
attributes	AttributeDescriptionList	☑	A list of the attributes to be returned from each entry which matches the search filter.

			Detailed description of AttributeDescriptionList is defined in RFC 2251 and RFC 2254

Table 17: LDAP search request

The results of the search attempted by the server upon receipt of a Search Request are returned in Search Response, which is LDAP message containing SearchResultEntry.

SearchResultEntry contains the following attributes:

Attribute	Data Type	Mandatory?	Comments
objectName	LDAP Distinguished Name	☑	An LDAP Distinguished Name of object returned. Detailed description of LDAP Distinguished Name is defined in RFC 2253
attributes		☑	A list of attributes and their values returned
type	AttributeDescription	☑	Attribute name
vals	SET OF AttributeValue	☑	List of attribute values

Table 18: SearchResultEntry structure

The Search Request and Response structures are defined in RFC 2251. String representation of LDAP search filters is defined in RFC 2254.

6.1.3 LDAP related standards

LDAP Core standards and documents are provided in the list below:

- LDAPv3 Technical Specification (RFC 3377);
- LDAPv3 Protocol (RFC 2251);
- LDAPv3 Attribute Syntax Definitions (RFC 2252);
- LDAPv3 UTF-8 String Representation of Distinguished Names (RFC 2253);
- LDAPv3 String Representation of LDAP Search Filters (RFC 2254);
- LDAPv3 URL Format (RFC 2255);
- A Summary of the X.500(96) User Schema for use with LDAPv3 (RFC 2256);
- Authentication Methods for LDAP (RFC 2829);
- LDAPv3 Extension for Transport Layer Security (RFC 2830);
- IANA Considerations for LDAP (RFC 3383).

6.2 OCSP

The OCSP Responder of LP CSP is a component responsible for providing Online Certificate Status Protocol (OCSP) responses to customers and external parties requiring reliable certificate status validation according to RFC 2560.

These services of LP CSP are available online at

- http://ocsp.e-me.lv/responder.eme

Requests to the OCSP Responder of the LP CSP can be performed over HTTP and HTTPS (if caller authentication and data encryption is required).

The logical interface of OCSP Responder is defined in Table 19.

Method	Interface	Description
GetCertificateStatus	In: ▪ Certificate status request (binary blob) Out: ▪ Certificate status response (binary blob)	Actions: ▪ Authenticates user calling if caller is using the authenticated channel to access the OCSP. ▪ If caller is using channel for authenticated users but her credentials are not valid (e.g., certificate expired or not issued by LP CSP IS), *unauthorized* response message is returned. ▪ Validates certificate status request for following things defined in the RFC: ▪ If the request message is well formed; ▪ If the responder is configured to provide the requested service; ▪ the request contains the information needed by the responder. ▪ If validation fails, *malformedRequest* response is prepared and returned. ▪ If certificate status request is valid (in accordance to RFC), returns the response blob. ▪ If service fails and does not provide the response, returns *tryLater* response message. ▪ If there is a recoverable error during the operation of the responder, returns *internalError* response message.

Table 19: OCSP service interface definition

6.2.1 OCSP Request Format

This section specifies the ASN.1 specification for the OCSP request. The ASN.1 syntax imports terms defined in RFC 2459. For signature calculation, the data to be signed is encoded using the ASN.1 distinguished encoding rules (DER).

OCSP request (OCSPRequest) contains the following attributes:

Attribute	Data Type	Mandatory?	Comments
TBSRequest		☑	
• version	INTEGER { v1(0) }	☑	The version field (currently v1)

			describes the version of the OCSP request.
• requestorName	GeneralName	☐	Requestor name
• requestList	TBSRequest[]	☑	List of requests
○ hashAlgorithm	AlgorithmIdentifier	☑	The hash algorithm used for both these hashes, is identified in hashAlgorithm. One of the following values must be contained in algorithm identifier: • sha-1; • md5.
○ issuerNameHash	OCTET STRING	☑	issuerNameHash is the hash of the Issuer's distinguished name.
○ issuerKeyHash	OCTET STRING	☑	issuerKeyHash is the hash of the Issuer's public key.
○ serialNumber	CertificateSerialNumber	☑	serialNumber is the serial number of the certificate for which status is being requested.
○ singleRequestExtensions	Extensions	☐	The extensions field is a generic way to add additional information to the request in the future.
• requestExtensions	Extensions	☐	The extensions field is a generic way to add additional information to the request in the future.
optionalSignature		☐	
• signatureAlgorithm	AlgorithmIdentifier	☑	Algorithm OID. One of the following values must be contained in algorithm identifier:

			• sha-1; • md5.
• signature	BIT STRING	☑	Signature
• certs	Certificate[]	☐	List of certificates

Table 20: OCSP request

More details about OCSP request can be found in RFC 2560.

6.2.2 OCSP Response Format

This section specifies the ASN.1 specification for a confirmation response.

OCSP response (OCSPResponse) contains the following attributes:

Attribute	Data Type	Mandatory?	Comments
responseStatus	INTEGER	☑	An OCSP response at a minimum consists of a responseStatus field indicating the processing status of the prior request. If the value of responseStatus is one of the error conditions, responseBytes are not set. One of the following values must be contained in status: • 0 – successful • 1 – malformedRequest • 2 – internalError • 3 – tryLater • 5 – sigRequired • 6 – unauthorized
responseBytes		☐	The value for responseBytes consists of an OBJECT IDENTIFIER and a response syntax identified by that OID encoded as an OCTET STRING.
• responseType	OBJECT IDENTIFIER ::= { id-pkix-ocsp 1 }	☑	Constant
• response	OCTET STRING	☑	The value for response shall be the DER encoding of BasicOCSPResponse.
○ tbsResponseData	ResponseData (see definition	☑	

		below)		
	○ signatureAlgorithm	AlgorithmIdentifier	☑	Algorithm OID. One of the following values must be contained in algorithm identifier: • sha-1; • md5.
	○ signature	BIT STRING	☑	The value for signature shall be computed on the hash of the DER encoding ResponseData.
	○ certs	Certificate[]	☐	List of certificates.

Table 21: OCSP response

ResponseData structure contains the following attributes:

Attribute	Data Type	Mandatory?	Comments
version	INTEGER { v1(0) }	☑	The version field (currently v1) describes the version of the record.
responderID		☑	Contains byname or byKey.
• byname or	Name	☑	
• byKey	OCTET STRING	☑	SHA-1 hash of responder's public key (excluding the tag and length fields).
producedAt	GeneralizedTime	☑	
responses	SingleResponse[]	☑	List of responses.
• certID	CertID	☑	
• certStatus	INTEGER	☑	Returns certificate status. Contains good, revoked or unknown.
○ good or	NULL		
○ revoked or	RevokedInfo		
▪ revocationTime	GeneralizedTime	☑	
▪ revocationReason	CRLReason	☐	
○ unknown	NULL		
• thisUpdate	GeneralizedTime	☑	
• nextUpdate	GeneralizedTime	☐	
• singleExtensions	Extensions	☐	
responseExtensions	Extensions	☐	

Table 22: ResponseData structure

More details about OCSP response can be found in RFC 2560.

6.3 Web Access to CRLs

LP CSP maintains a Website that provides access to downloadable CRLs defined by the Certificate and CRL Profile (RFC 2459) standard. Both CRLs and Delta CRLs will be downloadable from the Website as files.

CRLs and Delta CRLs will be published both to the Website and LDAP directory.

6.3.1 CRL Distribution Point (CDP)

All CRL Distribution points which contain pointer to CRL in LDAP location will contain relative paths to that location to be compliant with RFC 3280 document.

The following table lists CRL Distribution Points which will be included in certificates issued by particular certification authority. Each CA will publish revocation lists which will be populated to public LDAP directory. For offline certification authorities it will be done manually by designed application with cooperation with appropriate personnel. For online servers it will be automatic process.

CRL Distribution Points (CDP) are provided in Table 23.

CDP for certificates issued by:	CRL Distribution Point (CDP)
Root CA	ldap://e-me.lv/CN=LP CSP Root CA,OU=Certificate Service Provider,O=Latvian Post,C=LV?certificateRevocationList?base?objectClass=certificationAuthority http://www.e-me.lv/CDP/LP CSP Root CA.crl
Policy CA 1	ldap://e-me.lv/CN=LP CSP Policy CA 1,OU=Certificate Service Provider,O=Latvian Post,C=LV?certificateRevocationList?base?objectClass=certificationAuthority http://www.e-me.lv/CDP/LP CSP Policy CA 1.crl
Policy CA 2	ldap://e-me.lv/CN=LP CSP Policy CA 2,OU=Certificate Service Provider,O=Latvian Post,C=LV?certificateRevocationList?base?objectClass=certificationAuthority http://www.e-me.lv/CDP/LP CSP Policy CA 2.crl
Issuing CA 1	
	Base CRL: ldap://e-me.lv/CN= LP CSP Issuing CA 1,OU=Certificate Service Provider,O=Latvian Post,C=LV?certificateRevocationList?base?objectClass=certificationAuthority http://www.e-me.lv/CDP/ LP CSP Issuing CA 1.crl Delta CRL: ldap://e-me.lv/CN= LP CSP Issuing CA 1,OU=Certificate Service Provider,O=Latvian Post,C=LV?deltaRevocationList?base?objectClass=certificationAuthority http://www.e-me.lv/CDP/ LP CSP Issuing CA 1+.crl
Issuing CA 2	Base CRL: ldap://e-me.lv/CN= LP CSP Issuing CA 2,OU=Certificate Service Provider,O=Latvian

	Post,C=LV?certificateRevocationList?base?objectClass=certificationAuthority http://www.e-me.lv/CDP/LP CSP Issuing CA 2.crl Delta CRL: ldap://e-me.lv/CN= LP CSP Issuing CA 2,OU=Certificate Service Provider,O=Latvian Post,C=LV?deltaRevocationList?base?objectClass=certificationAuthority http://www.e-me.lv/CDP/LP CSP Issuing CA 2+.crl

Table 23: CRL Distribution Points for CA

CRL lifetime for different certificate types is provided in Table 24.

CRL lifetime for certificates issued by:	CRL Type	Validity Period	Overlapping Period
Root CA	Full	3 month	1 week
Policy CA 1	Full	3 month	1 week
Policy CA 2	Full	3 month	1 week
Issuing CA 1	Full	7 days	3 days
	Delta	24 h	3 days
Policy CA 1	Full	7 days	3 days
	Delta	24 h	3 days

Table 24: CRL lifetime

6.4 TSP

Time stamping services are based upon TSP (Time-Stamp protocol) defined by the RFC 3161 standard. Authentication for the time stamping service is implemented by wrapping the TSA request into PKCS#7 package signed using customer's non-repudiation certificate (the same used for signing the document to be time stamped).

As the first message of this mechanism, the requesting entity requests a time-stamp token by sending a request to the Time Stamping Authority. As the second message, the Time Stamping Authority responds by sending a response to the requesting entity.

The logical interface of the TSA service is defined in Table 25.

Method	Interface	Description
RequestTimestamp	In: ■ Timestamp request (binary blob) Out: ■ Timestamp response (binary blob)	Actions: ■ Authenticates user calling. ■ If client cannot be authenticated or the credentials are not valid returns PKIStatus *rejection* and PKIFailureInfo *badRequest*. ■ If validation should be performed ■ Parses and validates timestamp request to check basic request format: ▪ If the request message is well formed (if not returns PKIStatus *rejection* and PKIFailureInfo *badRequest)*; ▪ If recognized and supported hashing algorithm is used (if not returns PKIStatus *rejection* and PKIFailureInfo

Method	Interface	Description
		badAlg); ▪ If hash information corresponds to the specified algorithm (if not, returns PKIStatus *rejection* and PKIFailureInfo *badDataFormat*); ▪ If supported policy is specified by the caller (if not, returns PKIStatus *rejection* and PKIFailureInfo *unacceptedPolicy*); ▪ if any extension is specified (if any extension is used, returns PKIStatus *rejection* and PKIFailureInfo *unacceptedExtension*). ■ Returns the response blob. ■ If service fails, returns PKIStatus *rejection* and PKIFailureInfo *systemFailure*.

Table 25: TSA service interface definition

6.4.1 Time-Stamping Request Format

A time-stamping request (TimeStampReq) contains the following attributes:

Attribute	Data Type	Mandatory?	Comments
version	INTEGER { v1(1) }	☑	The version field (currently v1) describes the version of the Time-Stamp request.
messageImprint		☑	
• hashAlgorithm	AlgorithmIdentifier	☑	A hash algorithm OID. The hash algorithm indicated in the hashAlgorithm field should be a known hash algorithm (one-way and collision resistant). One of the following values must be contained in algorithm identifier: • sha-1; • md5.
• hashedMessage	OCTET STRING	☑	Hash value of the data to be time-stamped. Its length must match the length of the hash value for that algorithm (e.g., 20 bytes for SHA-1 or 16 bytes for MD5)
reqPolicy	OBJECT IDENTIFIER	☐	The reqPolicy field, if included, indicates the TSA policy under which the TimeStampToken should be provided.
nonce	INTEGER	☐	The nonce, if included, allows the client to verify the timeliness of the response when no local clock is available. The nonce is a large random number with a high probability that the client generates it only once (e.g., a 64 bit integer).
certReq	BOOLEAN	☐	If the certReq field is present and set to true, the TSA's public key certificate that is referenced by the ESSCertID identifier inside a

			SigningCertificate attribute in the response must be provided by the TSA in the certificates field from the SignedData structure in that response. That field may also contain other certificates. If the certReq field is missing or if the certReq field is present and set to false then the certificates field from the SignedData structure must not be present in the response.
extensions	Extensions	☐	The extensions field is a generic way to add additional information to the request in the future. Extensions are defined in [RFC 2459].

Table 26: Time-stamping request

More details about time-stamping request can be found in RFC 3161.

6.4.2 Time-Stamping Response Format

A time-stamping response (TimeStampResp) contains the following attributes:

Attribute	Data Type	Mandatory?	Comments
status		☑	
• status	INTEGER	☑	The status is based on the definition of status in section 3.2.3 of [RFC2510]. When the status contains the value zero or one, a TimeStampToken must be present. When status contains a value other than zero or one, a TimeStampToken must not be present. One of the following values must be contained in status: • 0 – granted • 1– grantedWithMods • 2 – rejection • 3 – waiting • 4 – revocationWarning • 5 – revocationNotification
• statusString	PKIFreeText	☐	The statusString field of PKIStatusInfo may be used to include reason text such as "messageImprint field is not correctly formatted".
• failInfo	BIT STRING	☐	When the TimeStampToken is not present, the failInfo indicates the reason why the time-stamp request was rejected and may be one of the following values: • 0 – badAlg • 2 – badRequest • 5 – badDataFormat

			• 14 – timeNotAvailable • 15 – unacceptedPolicy • 16 – unacceptedExtension • 17 – addInfoNotAvailable • 25 – systemFailure
timeStampToken		☐	
• contentType	OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) ct(1) 4}	☑	eContentType is an object identifier that uniquely specifies the content type.
• content	TSTInfo (see definition below)	☑	Content itself, carried as an octet string. It contains DER-encoded value of TSTInfo structure.

Table 27: Time-stamping response

TSTInfo structure contains the following attributes:

Attribute	Data Type	Mandatory?	Comments
version	INTEGER { v1(1) }	☑	The version field (currently v1) describes the version of the time- stamp token.
policy	TSAPolicyId	☑	The policy field must indicate the TSA's policy under which the response was produced. If a similar field was present in the TimeStampReq, then it must have the same value, otherwise an error (unacceptedPolicy) must be returned. This policy may include the following types of information: • The conditions under which the time-stamp token may be used; • The availability of a time-stamp token log, to allow later verification that a time-stamp token is authentic.
messageImprint	MessageImprint	☑	Must have the same value as the similar field in TimeStampReq
serialNumber	INTEGER	☑	Time-Stamping users must be ready to accommodate integers up to 160 bits. It MUST be unique for each TimeStampToken issued by a given TSA (i.e., the TSA name and serial number identify a unique TimeStampToken).
genTime	GeneralizedTime	☑	genTime is the time at which the time-stamp token has been created by the TSA.

accuracy		☐	accuracy represents the time deviation around the UTC time contained in GeneralizedTime.
• seconds	INTEGER	☐	Seconds
• millis	INTEGER (1..999)	☐	Milliseconds
• micros	INTEGER (1..999)	☐	Microseconds
ordering	BOOLEAN	☐	Default = FALSE
nonce	INTEGER	☐	Must be present if the similar field was present in TimeStampReq. In that case it MUST have the same value.
tsa	GeneralName	☐	The purpose of the tsa field is to give a hint in identifying the name of the TSA.
extensions	Extensions	☐	extensions is a generic way to add additional information in the future. Extensions is defined in [RFC 2459]

Table 28: TSTInfo structure

More details about time-stamping response can be found in RFC 3161.

6.5 Document Validation Service

For external systems, e.g. e-services providers, server-side document validation interface is provided. This interface is built as XML Web service.

6.5.1 Document Submission

The calling application initiates communication with the Document Validation Service by calling Submit method. The Document Validation Service returns one of the following types of replies: Acknowledgement, Response or Error. If an Acknowledgement is returned, the client application must keep polling the Document Validation Service until it receives either a Response or Error. This typically means that the client application and the Document Validation Service will keep exchanging pairs of polls and acknowledgements until either a response or error state is reached. Once a Response has been received the client application must then call a Dispose method for the deletion of the original validation result message. Only when the client application has received a Response message back from the Document Validation Service Dispose method, can a transaction be considered to be finished.

LP CSP provides server-side document validation XML Web Services only for authenticated clients of the Post CSP IS. For caller authentication certificate based tokens according to WS-Security standard will be used.

Calling applications submits array of documents to the Document Validation Service by calling Submit method. Individual document record consists of original file name and binary file content.

Logical interface of the document validation service is defined in Table 29.

Method	Interface	Description
SubmitDocuments	In: ▪ Document[] ▪ File name	Called by external systems wishing to validate the documents in LP CSP IS. Actions:

Method	Interface	Description
	▪ File content ▪ SignedResponseRequired (default false). ▪ NotificationRequired (default false). Out: ▪ Status ▪ ErrorInformation ▪ ValidationRequestInfo ▪ ValidationRequestId ▪ PollInterval ▪ ValidationResult ▪ TimeStamp ▪ Signature ▪ ValidationResults	▪ Validates the certificate used for starting HTTPS session by validating the certificate. ▪ If client is not authenticated, returns authentication error. ▪ Validates the size of the validation request (total size of all documents in the request). If it exceeds certain configured limit, returns Status = Rejected. ▪ Validates structure of the documents against the document format and supplied template. Validation is done using EDoc Library. ▪ If error found in all Documents in the supplied document list and SignedResponseRequired is false, returns validation error information defining that supplied documents are not valid according to template or document format. Status = Done. Timestamp and signature in validation response are empty in this case. ▪ Sends responses back to the caller. ▪ If service fails, returns Status = Error with ErrorInformation stating that service temporarily is not available.
GetValidationRequestStatuses	In: ▪ DateFrom ▪ DateTo Out: ▪ Status ▪ ErrorInformation ▪ ValidationRequestInfo[] ▪ ValidationRequestId ▪ ValidationStatus ▪ ConsolidatedValidationResult ▪ DateSubmitted ▪ RequestDocumentCount	Actions: ▪ Validates the certificate used for starting HTTPS session by validating the certificate. ▪ If client is not authenticated, returns authentication error. ▪ Gets request information for given subject with submission date in given date range. ▪ If number of filtered items exceeds preconfigured limit N, returns only first N items and Status = SuccessHasMore ▪ In case when limit is not violated, returns Status = Success. ▪ Returns results if no error encountered. ▪ If service fails, returns Status = Error with ErrorInformation stating that service temporarily is not available.
Poll	In: ▪ ValidationRequestId Out: ▪ Status ▪ ErrorInformation ▪ ValidationResult ▪ TimeStamp ▪ Signature ▪ ValidationResults	Actions: ▪ Validates the certificate used for starting HTTPS session by validating the certificate. ▪ If client is not authenticated, returns authentication error. ▪ Checks if documents have been already validated ▪ If the validation result is ready ▪ Returns the ValidationResult and Status = Done ▪ If the result is not ready ▪ Returns Status = NotComplete ▪ If the entry with specified SubjectId and ValidationRequestId not found ▪ Returns Status = NotFound ▪ Sends responses back to the caller. Specifics:

Method	Interface	Description
		▪ If service fails, returns Status = Error with ErrorInformation stating that service temporarily is not available.
Dispose	In: ▪ ValidationRequestId Out: ▪ Status ▪ ErrorInformation	Actions: ▪ Validates the certificate used for starting HTTPS session by validating the certificate. ▪ If client is not authenticated, returns authentication error. ▪ Removes the validation request and associated response (if response already exists) from the database ▪ If matching request with given ValidationRequestId was found and removed ▪ Returns Status = Done ▪ If record not found ▪ Returns Status = NotFound ▪ Sends responses back to the caller. Specifics: ▪ If service fails, returns Status = Error with ErrorInformation stating that service temporarily is not available.

Table 29: Document Validation Web Service interface definition

The Document Validation Service may delete responses after a significant period of time has elapsed since the client last sent any type of message relating to that validation request.

6.5.1.1 Protocol States

There are five major protocol states which the conversation between the client and the Document Validation Service can enter:

- **Submission**
  Entered as soon as the client first initiates communication with the Document Validation Service;
- **Polling**
  Entered once the client has received a Acknowledgement message from the Document Validation Service. The client must keep polling the Document Validation Service until a Response is received;
- **Response Received**
  Entered once the client has received a Response from the Document Validation Service;
- **Delete Response**
  Entered once the client calls Dispose method to delete the response;
- **Error State**
  Entered if the client receives any form of error response from the Document Validation Service.

6.5.2 Usage limitations of signature validation

Signature validation service is available either through e-Signing application or through Web CSP. There are following usage limitations for both types of signature validation.

Signature validation	LP CSP subjects with valid LP CSP issued authentication	Other people	Comment

check	certificate				
	e-Signing Application	Web CSP Application	e-Signing Application	Web CSP Application	
Document format check (general conditions)	Yes	Yes	Yes	No	Major check
Document format (metadata for specific document type – plug-in registered within LP CSP)	Yes	Yes	Yes	No	Minor check
Digital signature integrity check	Yes	Yes	Yes	No	Major check
Certificate status check (LP CSP issued certificates)	Yes (when online – OCSP, when offline – cached CRL)	Yes (OCSP)	Yes	No	Major check
Certificate status check (certificates issued by other CSP)	Yes (CRLs only)	Yes (CRLs only)	Yes (CRLs only)	No	Major check
Timestamp integrity and validity check (LP CSP issued TS)	Yes	Yes	Yes	No	Major check
Timestamp integrity check (TS issued by other CSP)	Yes	Yes	No, because requires authentication	No, because requires authentication	Minor check
Timestamp validity check (other TSA issued TS)	No	No	No	No	Minor check

Table 30: Availability of signature validation checks

APPENDIX A OPEN XML CONTENT TYPES SCHEMA

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema
targetNamespace="http://schemas.openxmlformats.org/package/2006/content-
types" elementFormDefault="qualified" attributeFormDefault="unqualified"
blockDefault="#all" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.openxmlformats.org/package/2006/content-types"
xmlns:odoc="http://schemas.microsoft.com/office/internal/2005/internalDocum
entation">
  <xsd:element name="Types" type="CT_Types">
    <xsd:annotation>
      <xsd:documentation>Part Content Types</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:complexType name="CT_Types">
    <xsd:annotation>
      <xsd:documentation>Types Complex Type</xsd:documentation>
    </xsd:annotation>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="Default" type="CT_Default">
        <xsd:annotation>
          <xsd:documentation>Default Content Type</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="Override" type="CT_Override">
        <xsd:annotation>
          <xsd:documentation>Content Type Override</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="CT_Default">
    <xsd:annotation>
      <xsd:documentation>Default Content Type
definition</xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="Extension" type="ST_Extension" use="required" />
    <xsd:attribute name="ContentType" type="ST_ContentType" use="required"
/>
  </xsd:complexType>
  <xsd:complexType name="CT_Override">
    <xsd:annotation>
      <xsd:documentation>Override Complex Type</xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="ContentType" type="ST_ContentType" use="required"
/>
    <xsd:attribute name="PartName" type="xsd:anyURI" use="required" />
  </xsd:complexType>
  <xsd:simpleType name="ST_ContentType">
    <xsd:restriction base="xsd:string">
      <xsd:whiteSpace value="collapse" />
      <xsd:pattern
value="((([^\p{Cc}□\(\)&amp;lt;&amp;gt;@,;:\\&amp;quot;/\[\]\?=\{\}\s\t]+))/(([^\p{Cc}□
\(\)&amp;lt;&amp;gt;@,;:\\&amp;quot;/\[\]\?=\{\}\s\t]+))((\s+)*;(\s+)*((([^\p{Cc}□\(\)&
lt;&amp;gt;@,;:\\&amp;quot;/\[\]\?=\{\}\s\t]+))=(([^\p{Cc}□\(\)&amp;lt;&amp;gt;@,;:\\&amp;quot;
/\[\]\?=\{\}\s\t]+)|(&amp;quot;(([^\p{Cc}□&amp;quot;\n\r]|(\s+))|(\\[\p{IsBasicLati
n}]))*&amp;quot;))))*)" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ST_Extension">
```

```
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="([!$&amp;'\(\)\*\+,:=]|(%[0-9a-fA-F][0-9a-fA-
F])|[:@]|[a-zA-Z0-9\-_~])*" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

APPENDIX B OPEN XML RELATIONSHIPS SCHEMA

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema
targetNamespace="http://schemas.openxmlformats.org/package/2006/relationshi
ps" blockDefault="#all" elementFormDefault="qualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:rs="http://schemas.openxmlformats.org/package/2006/relationships"
xmlns:odoc="http://schemas.microsoft.com/office/internal/2005/internalDocum
entation">
  <xsd:element name="Relationships" type="rs:CT_Relationships">
    <xsd:annotation>
      <xsd:documentation>Package Relationships</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:complexType name="CT_Relationships">
    <xsd:annotation>
      <xsd:documentation>Relationships XML Element</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref="rs:Relationship" minOccurs="0"
maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>Package Relationship</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="Relationship" type="rs:CT_Relationship">
    <xsd:annotation>
      <xsd:documentation>Relationship XML Element</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:complexType name="CT_Relationship">
    <xsd:annotation>
      <xsd:documentation>Package Relationship Type</xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="TargetMode" type="rs:ST_TargetMode" use="optional"
default="Internal">
      <xsd:annotation>
        <xsd:documentation>Relationship Target Mode</xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="Target" type="xsd:anyURI" use="required">
      <xsd:annotation>
        <xsd:documentation>Relationship XML Attributes
Reference</xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="Type" type="xsd:anyURI" use="required">
      <xsd:annotation>
        <xsd:documentation>Relationship XML Attributes
Reference</xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="Id" type="xsd:ID" use="required">
      <xsd:annotation>
        <xsd:documentation>Relationship XML Attributes
Reference</xsd:documentation>
      </xsd:annotation>
```

```
    </xsd:attribute>
  </xsd:complexType>
  <xsd:simpleType name="ST_TargetMode">
    <xsd:annotation>
      <xsd:documentation>Relationship XML TargetMode
type</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="External">
        <xsd:annotation>
          <xsd:documentation>Relationship XML Attributes
Reference</xsd:documentation>
        </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration value="Internal">
        <xsd:annotation>
          <xsd:documentation>Relationship XML Attributes
Reference</xsd:documentation>
        </xsd:annotation>
      </xsd:enumeration>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

APPENDIX C OPEN XML CORE PROPERTIES SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://schemas.openxmlformats.org/package/2006/metadata/co
re-properties"
xmlns="http://schemas.openxmlformats.org/package/2006/metadata/core-
properties" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:dcterms="http://purl.org/dc/terms/" elementFormDefault="qualified"
blockDefault="#all">

      <!--
schemaLocation="http://dublincore.org/schemas/xmls/qdc/2003/04/02/dc.xsd" -
->
      <xs:import namespace="http://purl.org/dc/elements/1.1/" />

      <!--
schemaLocation="http://dublincore.org/schemas/xmls/qdc/2003/04/02/dcterms.x
sd" -->
      <xs:import namespace="http://purl.org/dc/terms/" />

      <xs:element name="coreProperties" type="CT_coreProperties">
        </xs:element>

    <xs:complexType name="CT_coreProperties">
        <xs:all>
            <xs:element ref="dc:creator" minOccurs="0" maxOccurs="1">
                </xs:element>
            <xs:element ref="dcterms:created" minOccurs="0" maxOccurs="1">
                </xs:element>
            <xs:element ref="dc:identifier" minOccurs="0" maxOccurs="1">
                </xs:element>
            <xs:element name="contentType" minOccurs="0" maxOccurs="1"
type="xs:string">
                </xs:element>
            <xs:element ref="dc:title" minOccurs="0" maxOccurs="1">
                </xs:element>
            <xs:element ref="dc:subject" minOccurs="0" maxOccurs="1">
                </xs:element>
            <xs:element ref="dc:description" minOccurs="0" maxOccurs="1">
                </xs:element>
            <xs:element name="keywords" minOccurs="0" maxOccurs="1"
type="xs:string">
                </xs:element>
            <xs:element ref="dc:language" minOccurs="0" maxOccurs="1">
                </xs:element>
            <xs:element name="category" minOccurs="0" maxOccurs="1"
type="xs:string">
                </xs:element>
            <xs:element name="version" minOccurs="0" maxOccurs="1"
type="xs:string">
                </xs:element>
            <xs:element name="revision" minOccurs="0" maxOccurs="1"
type="xs:string">
                </xs:element>
            <xs:element name="lastModifiedBy" minOccurs="0" maxOccurs="1"
type="xs:string">
                </xs:element>
```

```
            <xs:element ref="dcterms:modified" minOccurs="0" maxOccurs="1">
                </xs:element>
            <xs:element name="lastPrinted" minOccurs="0" maxOccurs="1"
type="xs:dateTime">
                </xs:element>
            <xs:element name="contentStatus" minOccurs="0" maxOccurs="1"
type="xs:string">
                </xs:element>
        </xs:all>
    </xs:complexType>
</xs:schema>
```

APPENDIX D EDOC MANIFEST SCHEMA

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="Manifest"
  targetNamespace="http://schemas.microsoft.com/edoc/2006/manifest"
  elementFormDefault="unqualified"
  xmlns="http://schemas.microsoft.com/edoc/2006/manifest"
  xmlns:mstns="http://schemas.microsoft.com/edoc/2006/manifest"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Manifest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DocumentType" type="xs:string" />
        <xs:element name="TemplateLocation" type="xs:string" />
        <xs:element name="Format" type="xs:string" />
        <xs:element name="Version" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

APPENDIX E EDOC TEMPLATE SCHEMA

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="EdocTemplate"
targetNamespace="http://schemas.microsoft.com/edoc/2006/EdocTemplate.xsd"
elementFormDefault="qualified"
xmlns="http://schemas.microsoft.com/edoc/2006/EdocTemplate.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="templateInfo" type="TemplateInfo">
  </xs:element>
  <xs:simpleType name="AllowedTypes">
    <xs:restriction base="xs:string">
      <xs:enumeration value="text" />
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="Choice">
    <xs:sequence>
      <xs:element name="value" type="xs:string" minOccurs="1"
maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="allowUserValues" type="xs:boolean" default="false"
use="optional" />
  </xs:complexType>
  <xs:element name="EdocTemplate">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="info" type="TemplateInfo" maxOccurs="1"
minOccurs="1" nillable="false" />
        <xs:element name="namespace" type="NamespaceType"
maxOccurs="unbounded" minOccurs="0">
        </xs:element>
        <xs:element name="properties" type="PropertyList" maxOccurs="1"
minOccurs="1">
        </xs:element>
        <xs:element name="restrictions" maxOccurs="1" minOccurs="0"
type="TemplateRestrictions" nillable="true">
        </xs:element>
        <xs:element name="signatures" type="SignatureTypes" nillable="true"
minOccurs="0" maxOccurs="1">
        </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required" />
      <xs:attribute name="sectionName" type="xs:string" default="Extended"
/>
      <xs:attribute name="url" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:complexType name="TemplateInfo">
    <xs:sequence>
      <xs:element name="organization" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="description" type="xs:string" maxOccurs="1"
minOccurs="1" />
      <xs:element name="version" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="TemplateProperty">
    <xs:sequence>
      <xs:element name="choice" nillable="false" type="Choice"
maxOccurs="1" minOccurs="0">
```

```
      </xs:element>
    </xs:sequence>
    <xs:attribute name="name" type="xs:NCName" use="required" />
    <xs:attribute name="mandatory" type="xs:boolean" use="optional"
default="false" />
    <xs:attribute name="dataType" type="AllowedTypes" use="optional"
default="text">
    </xs:attribute>
    <xs:attribute name="description" type="xs:string" use="required" />
    <xs:attribute name="readonly" type="xs:boolean" />
    <xs:attribute name="namespace" type="xs:string" />
    <xs:attribute name="defaultValue" type="xs:string" />
  </xs:complexType>
  <xs:complexType name="PropertyList">
    <xs:sequence>
      <xs:element name="property" type="TemplateProperty" minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="TemplateRestrictions">
    <xs:sequence>
      <xs:element name="maxSignatures" type="xs:int" default="0"
minOccurs="0" maxOccurs="1" nillable="false" />
      <xs:element name="maxDocuments" type="xs:int" nillable="false"
default="0" minOccurs="0" maxOccurs="1" />
      <xs:element name="maxSizeMb" type="xs:int" nillable="false"
default="0" minOccurs="0" maxOccurs="1" />
      <xs:element name="minSignatures" type="xs:int" nillable="false"
default="0" minOccurs="0" maxOccurs="1">
      </xs:element>
      <xs:element name="minDocuments" type="xs:int" nillable="false"
default="0" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SignatureTypes">
    <xs:sequence>
      <xs:element name="customTypes" type="SignatureType" minOccurs="0"
maxOccurs="unbounded">
      </xs:element>
    </xs:sequence>
    <xs:attribute name="simpleSignatureAllowed" type="xs:boolean"
default="true" use="optional" />
  </xs:complexType>
  <xs:complexType name="SignatureType">
    <xs:sequence>
      <xs:element name="properties" type="PropertyList" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="basic_sig_allowed" type="xs:boolean" use="optional"
default="true" />
    <xs:attribute name="timestamped_sig_allowed" type="xs:boolean"
default="true" use="optional" />
    <xs:attribute name="verified_sig_allowed" type="xs:boolean"
default="true" use="optional" />
  </xs:complexType>
  <xs:complexType name="NamespaceType">
    <xs:sequence>
    </xs:sequence>
    <xs:attribute name="prefix" type="xs:string" />
    <xs:attribute name="url" type="xs:string" />
  </xs:complexType>
```

```
</xs:schema>
```

APPENDIX F EDOC CORE TEMPLATE

```
<?xml version="1.0" encoding="utf-8" ?>
<t:EdocTemplate
  id="URN:MICROSOFT:LV:EDOCTEMPLATE:CORE"
  sectionName="Core"
  url="http://schemas.microsoft.com/edoc/2006/Core.template.xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:t="http://schemas.microsoft.com/edoc/2006/EdocTemplate.xsd"

xsi:schemaLocation="http://schemas.microsoft.com/edoc/2006/EdocTemplate.xsd
..\schemas\EdocTemplate.xsd">
  <t:info>
    <t:organization>Latvian Post</t:organization>
    <t:description>eDoc template - core properties</t:description>
    <t:version>1</t:version>
  </t:info>
  <t:namespace prefix="dc" url="http://purl.org/dc/elements/1.1/"/>
  <t:namespace prefix="opc"
url="http://schemas.openxmlformats.org/package/2006/metadata/core-
properties"/>
  <t:properties>
    <t:property name="Created" dataType="text" description="Creation
date/time" mandatory="true" readonly="true" namespace="dc"
defaultValue="01-Sep-2006 00:00:00"/>
    <t:property name="Language" dataType="text" description="Document
language" mandatory="true" readonly="true" namespace="dc"
defaultValue="English"/>
    <t:property name="Last_Modified_By" dataType="text" description="Last
Modified By" mandatory="true" readonly="true" namespace="opc"
defaultValue="user"/>
    <t:property name="Modified" dataType="text" description="Last
modification date" mandatory="true" readonly="true" namespace="dc"
defaultValue="01-Sep-2006 00:00:00"/>
    <t:property name="Revision" dataType="text" description="Revision
Number" mandatory="false" readonly="true" namespace="opc"
defaultValue="1"/>
    <t:property name="Title" dataType="text" description="Title"
mandatory="false" namespace="dc"  defaultValue="Default Title"/>
    <t:property name="Version" dataType="text" description="Version"
mandatory="false" namespace="dc" defaultValue="1"/>
    <t:property name="Creator" dataType="text" description="Author name"
mandatory="false" namespace="dc" defaultValue="user"/>
  </t:properties>
</t:EdocTemplate>
```

APPENDIX G EDOC BASE TEMPLATE

```
<?xml version="1.0" encoding="utf-8" ?>
<t:EdocTemplate id="URN:MICROSOFT:LV:EDOCTEMPLATE:BASE"
  sectionName="Base"
  url="http://schemas.microsoft.com/edoc/2006/Base.template.xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:t="http://schemas.microsoft.com/edoc/2006/EdocTemplate.xsd"

xsi:schemaLocation="http://schemas.microsoft.com/edoc/2006/EdocTemplate.xsd
..\Schemas\EdocTemplate.xsd">
  <t:info>
    <t:organization>Latvian Post</t:organization>
    <t:description>eDoc template - base properties</t:description>
    <t:version>1</t:version>
  </t:info>
  <t:namespace prefix="edoc"
url="http://schemas.microsoft.com/edoc/2006/metadata"/>
  <t:properties>
    <t:property name="eDoc_Format" dataType="text" description="eDoc Format
version" mandatory="true" readonly="true" namespace="edoc"
defaultValue="0.0.1"/>
  </t:properties>
</t:EdocTemplate>
```

APPENDIX H EDOC EXTENDED TEMPLATE

```
<?xml version="1.0" encoding="utf-8" ?>
<t:EdocTemplate
  id="URN:MICROSOFT:LV:EDOCTEMPLATE:EXTENDED"
  sectionName="Papildus īpašības"
  url="http://schemas.microsoft.com/edoc/2006/Extended.template.xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:t="http://schemas.microsoft.com/edoc/2006/EdocTemplate.xsd"

xsi:schemaLocation="http://schemas.microsoft.com/edoc/2006/EdocTemplate.xsd
EdocTemplate.xsd">
  <t:info>
    <t:organization>Latvian Post</t:organization>
    <t:description>Tukša veidne</t:description>
    <t:version>1</t:version>
  </t:info>
      <t:namespace prefix="extended"
url="http://schemas.microsoft.com/edoc/2006/extended-section"/>
  <t:properties>
  </t:properties>
</t:EdocTemplate>
```

APPENDIX I EDOC SECTION REFERENCE

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="SectionReference"
targetNamespace="http://schemas.microsoft.com/edoc/2006/SectionReference.xs
d" elementFormDefault="qualified"
xmlns="http://schemas.microsoft.com/edoc/2006/SectionReference.xsd"
xmlns:mstns="http://schemas.microsoft.com/edoc/2006/SectionReference.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="section_reference">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="relative_path" type="xs:string" maxOccurs="1"
minOccurs="1">
        </xs:element>
        <xs:element name="hash" type="xs:string" maxOccurs="1"
minOccurs="1" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

APPENDIX J EDOC SIGNATURE PROPERTIES

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="signature_properties"
targetNamespace="http://schemas.microsoft.com/edoc/2006/signature_propertie
s.xsd" elementFormDefault="qualified"
xmlns="http://schemas.microsoft.com/edoc/2006/signature_properties.xsd"
xmlns:mstns="http://schemas.microsoft.com/edoc/2006/signature_properties.xs
d" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="signature_properties">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="property_type" maxOccurs="unbounded"
minOccurs="0">
          <xs:complexType>
            <xs:sequence>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" />
            <xs:attribute name="value" type="xs:string" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="level_type">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Basic" />
            <xs:enumeration value="Timestamped" />
            <xs:enumeration value="Verified" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="sigType" type="xs:string" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```