LTANS	A. Jerman Blazic	
Internet-Draft	SETCCE	
Intended status: Standards Track	P. Sylvester	
Expires: January 14, 2010	Groupe ON-X - EdelWeb Project	
	C. Wallace	
	Cygnacom Solutions	
	July 13, 2009	

TOC

Long-term Archive Protocol (LTAP)
draft-ietf-ltans-ltap-08

Status of this Memo

Copyright Notice

Abstract

This document describes a service operated as a trusted third party to securely archive electronic documents called a long-term archive service (LTA). We describe an architecture framework and a protocol allowing clients to interact with such a service. Bindings to concrete transport and security protocol layers are given.

Table of Contents

---

1. Introduction and Rationale

TOC

The possibility of long term conservation of documents has always been a key facture for cultures. Until recently, there were only few different technologies and procedures to achieve these goals, and they were used over long periods allowing to determine their quality. The technologies mostkly used up to day include paper and micro-film. Since about half a century, technologies to store information in new forms and on new media, for example digitally encoded on electronic media. The amount of data created since has been augmented enormously due the simplicity of aquisition, treatment, and access.
Contrary to the past, the simplicity of treatment often relies on the usage and availability of tools to access to stored information, and these tools are not rarely tied to specific storage technologies and data formats, and some tools and technologies are treated as proprietary. The innovation rate is rather high, and ti is not uncommon to see storage technolgies become obsolete or unusable after a decade of existance. Combined the exponential growth of data (not necessarily information) that is created and not organised properly. An equivalent amount of medium and long term archiving procedures as for the paper

world has not been developped. This complex situation results in large amount of data cimetaries, i.e. the loss of very valuable information. We note that these problems of the electronic world are not totally specific to it; in the past 200 years we have seen for example a catastophy in paper based conservation with the emergence of chemical processes and environmental polution which have made certain types of paper very unstable which had created unforeseen problems in archiving. Conservation of documents is important, one can even say that appropriate conservation rules are a prerequisite for data to become a document. Conservation has several aspects, e.g., duration or accessibility, which vary based on the nature of the document. For example, a document may be conserved for a certain period and may be destroyed after that, or its lifetime may be extended when the document becomes part of a conflict. Also, documents may become part of historical archives. A document may be accessible on a public or restricted basis to a set of potentially interested or authorized entities. It must be able to determine the quality of the stored information, and to be able to migrate the physical and logical support. It is also necessary for a very long term to foresee the possibility to migrate data formats without loss of authenticity. Conservation of documents needs to be treated at several layers, there are at least legal, organisational, semantical, syntactical, technological, physical layers and even security, mental, moral, ethical or philosophical aspects, etc. This specification of this document are situated largely in the technical and syntactical layers and provide a security and stability layer for stored douments based on the use of a specialized service for conservation of electronic documents called long-term archive service (LTA). The service creates and makes available enough information to demonstrate the existence, integrity and authenticity of electronic data over any period of time. In other words, the service assumes the responsibility to retrieve and, optionally, store data for conservation, create and store evidence to guarantee data integrity and completeness, and to maintain accessibility of data and evidence created.

This document describes a protocol for interacting with an LTA service. The document contains only description of a general request and response structure, and a detailed protocol description concerning access to an LTA service. Other specifications and descriptions, e.g. a framework protocol containing mappings to transport and security services, are addressed elsewhere. The protocol is intended to be used in client-server architecture, where client is simply an end user (a physical user or another service) and the server as an LTA service provider. In the sequel we often imit the word service or server and refer to the provider functionality or implementation simply as LTA.

The process of replacing paper based workflow and document handling which is also known as 'dematerialization' ignores to a certain degree the requirements for long-term stability of documents. Document conservation is generally performed by specialized services. For electronic formats it is proposed to use similar approaches, while

maintaining the distance of technical characteristics (paper versus electronic). Conservation might be taken out from other workflow activities, while the same procedures (evidence creation) might be used for any milestone in an electronic document lifecycle (e.g. version marking).
Since conservation of documents created by one entity is only necessary if there is a potential entity to which the document may be presented at some time, the conservation service (LTA) acts as a trusted third party for those two entities. The main role of an LTA is to generate and provide enough information for archived data existence in time, integrity and authenticity demonstration over long periods of time. Provision of data storage services is optional and may be assured by supportive infrastructure (e.g. database or document storage/management system).
Conservation is more that just storing a document. Not only, but in particular when the life time is very long, appropriate measures to ensure the integrity and provable or, a bit more realistic, highly plausible authenticity of the document need to be implamented. This aspect is handled by this specification. Sometimes, complete transfer of the document information to a new physical support has to be done like transformations from marble to paper or paper to microfilm. The need for such transformation makes the definition of what constitutes the actual document somewhat difficult, in particular it should be done independantly of the support, and even independantly of a current format of the document information. Transformation of documents are out of scope of this specification besides the obvious possibility of storing all formats of a document and assertions about the transformations.
Defined data structures are presented in XSD and ASN1 forms. Most XSD elements have been automatically generated from the ASN.1. XER encoded data are compatible with data structures encoded according to the XSD.

1.1. Requirements notation

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.).

---

2. Framework

TOC

This chapter describes a general framework for secure exchange of request and response messages between an archive client and archive

server, e.g. an LTA. It provides a high level outline and identifies common and external aspects from the concrete protocol data units.

2.1. Functional Overview

TOC

A conservation service or long-term archive (LTA) consists of several functional blocks. Some of these blocks are not considered as they present basic infrastructure, such as the communication network, storage device, data management, etc. Instead, an LTA implements the archive interaction protocol as defined by this specification (LTAP) and manages archive objects (logically interpreted as packages of archive data and conservation attributes) and evidence records [RFC4998] (Gondrom, T., Brandner, R., and U. Pordesch, “Evidence Record Syntax (ERS),” August 2007.). An LTA is a part of a general archive service that provides evidence used to demonstrate the existence of an archived data object at a given time and the integrity of the archived data object since that time. The LTA is the primary part tasked with creating and delivering conservation attributes for archived data. [RFC4810] (Wallace, C., Pordesch, U., and R. Brandner, “Long-Term Archive Service Requirements,” March 2007.) defines the services that must be provided by an LTA. A principal function of the LTA is to generate or obtain evidence information for (archive) data submited. An LTA may accept and store data for which it generates (or acquires from another service) and maintains evidence inforamtion. Alternatively, it may simply act as an evidence or information service without data storage capabilities (it relies upon other services for storage of the archived data). Evidence generated and maintained by an LTA addresses the problems of long-term integrity and temporal existence.
Archive objects are the central logical structures defined by the LTA and maintained on a long-term basis. They are atomic elements of an LTA service consisting of three logical parts:

- Archive data (including metadata or other related data) entering the LTA using the interaction protocol,
- Archive process-related meta or binding information, and
- Evidence information

The archive data may contain data of any type, e.g., raw data, signed data, encrypted data or time stamped data as defined by [RFC4810] (Wallace, C., Pordesch, U., and R. Brandner, “Long-Term Archive Service Requirements,” March 2007.). Archive data may be associated with additional data or attributes, e.g. meta information or digital signatures.
Data generated or collected by the LTA are archive process-related meta or binding information including demonstration information and evidence

information. Archive metadata may be needed to provide enough information for e.g. special (e.g., legal) purposes or validity demonstration purposes. Examples are complementary information to verify digital signatures or to attest their validity. The LTA collects meta or binding information directly from a user or some other entity (e.g. Certificate Authority). Such information may contain the data owner name, organization, location, etc.. Meta or binding information may be submitted using the LTAP.

Demonstration information is collected to demonstrate facts on the archive data. Such information may be digital signature reference information. The LTA may use external resources to collect such information, usually without user intervention. Evidence data is generated by the LTA or collected from an external resource, e.g. a time stamping authority. Evidence information is provided for all data: archive data submitted by a client and archive process-related data (including binding and demonstration information) collected by the LTA from the client or alternative resource.

The LTA performs perpetual maintenance of archive objects and associated metadata for the main purpose of demonstrating archive data existence in time and providing integrity information for the complete archiving time. Archive objects are periodically processed to provide long-term stability (e.g. by proof-reading, copying to new material, or performing time stamp renewal). This protocol specification make no assumptions about the details of such verification operations except that an implementation MUST specify how the outcome is reflected in the archive metadata of each object. The protocol provides for a simple means to initiate a verification on an archive object through a protocol action.

The LTAP protocol interprets the logical data structure to hold all needed information (including references) to build an archive object including archive data itself (or reference to archive data). The logical structure of the LTAP messages includes archive data, archiving process-related information and references together with request and processing information. Using LTAP, the LTA should have enough information to build and perform operations on an archive object or group of archive objects.

An LTA is a service that is responsible for preserving evidence data and/or data for long periods of time, as defined by [RFC4810] (Wallace, C., Pordesch, U., and R. Brandner, “Long-Term Archive Service Requirements,” March 2007.),. The service is accessible using the protocol defined in this document.

The protocol is intended to be used as well as core functionality available to customers of a service provider, as well as an internal protocol inside the LTA to access core building blocks as a black box. The latter usage is intended to permit the creating of security zones and auditable implementations. As a consequence, only a subset of functions defined in the protocol may be available to particular clients/customers of an LTA.

The protocol consists of two layers, the higher layer defines the available service functions and their mapping to encodings, the lower layer defines the rules for the exchange of protocol messages common for all all functions.
It is assumed that an LTA ensures the long-term availability of stored data and created evidence information, as necessary, and uses appropriate means to manage data and access rights. The details of these important features of an LTA are outside the scope of this specification.
The common high-level architecture consists of a protocol used to exchange requests and responses securely, potentially over different types of transport connections, to ensure the long-term validity of responses.
Clients and servers use one of several object types to build requests and responses. Data objects include raw (archive) data, request information, meta information, identification information and attestations.
Requests and responses are exchanged in a secure way responding to different security requirements, which may concern the security of the transport as well as the long-term validity of the data being exchanged.
The LTA is not a method for implementing something like a secure file system. In general, archived data are rarely accessed, restored or transferred. Thus, the archive operation is the most important one and performance is an important concern. In this case that client applications need frequent access to the data, they generally keep a copy of the data including evidence information, whose integrity can be compared from time to time or when requested.

2.2. Service functions of an LTA

TOC

The primary aim of this protocol is to enable a formal interaction between a client and an LTA. The result of the interaction is a verifiable attestation of procedures performed by an LTA (e.g. archive data plus evidence record). The format for data structures used to demonstrate integrity, i.e. to demonstrate that data has not undergone any transformations while in the care of the archive, is partially defined in other documents, namely in [RFC4998] (Gondrom, T., Brandner, R., and U. Pordesch, “Evidence Record Syntax (ERS),” August 2007.).
This specification does not place any requirements on the structure of archived data objects. However, it operates on elements that are derived from archive data objects (e.g. message imprints).
The LTA interface enables clients to perform at least the following operations in cooperation with an LTA:

> **ARCHIVE:**

> Submit data to an LTA and request creation of evidence information for data
>
> **EXPORT:** Retrieve data (including archive data, meta information and evidence information) from an LTA
>
> **DELETE:** Remove data and/or evidence information from an LTA.
>
> **VERIFY:** Determine the integrity and validity of LTA archived data.
>
> **STATUS:** Inform about the status of data.
>
> **LISTIDS:** Return a list of references to archived objects.

These operations MUST be implemented by an LTA service. There is a minimal profile for the parameter structures for transactions with a single server that does act as a final repository.
The operations are intended for different types of clients, including archive data owners but also entities that want to audit the service or are authorised to access to data. Since an LTA may restrict the access based on functions, a client MAY implement only the accessible subset of functions.
An LTA MAY propose additional services either by extending the operations by the means of additional parameters or by defining new operations like archive transfer, extension of lifetime, data splitting or relaying.
For example, often an extension of the initial lifetime of an object must be possible. With the core operations it is not possible to perform this in one operation. In order to do so, a client has to read the object and store it under a different service (not necessarily with the same provider). To extend the lifetime of an object, an EXPORT operation followed by an ARCHIVE operation has be used to transfer or copy the object to an LTA service having the required lifetime policy/ configuration.
Finally, after the transfer, the initial object may be deleted or not. The combination of such a sequence of operations into single one is an example of an extended service provided to clients.
Since it may not be desirable to make the data available to a potentially untrustworthy client, the combined operation could be implement using an extension to the core LTA functionality by allowing a reference to data as a parameter to the ARCHIVE function.

2.3. Transactions

TOC

The model for the exchange of LTAP requests and responses is borrowed from other environments and technologies like EDI, X.400 or ebXML. It's main characteristic is that exchanges for LTAP are conceptually

asynchronous. As a consequence, it is necessary to provide a mechanism to allow a client to determime that the LTA has received and processed a request. An LTAP exchange consists of sending a request and retrieving at least one of two different types of responses. A client initiates an archive service by submiting a request. This LTAP request consists of data to be archived and information related to the archive process. The process information may include client authorization, archive policy, service parameters, etc. The first type of response is a technical acknowledgement from the LTA that the request has been received and the process information has been accepted (or rejected). The second type of response is a statement from an LTA containing an indication of the outcome of the requested operation. This result (called an attestation) is, in general, a document with long-term validity allowing the client to reference the operation, and, in particular, to reference the data that has been preserved by the LTA. The asynchronous nature of the LTAP protocol is required by LTA operations, which may require a specific amount of time to perform, e.g. the archive operation needs to safely store the data and to produce evidence information.

The possibility to deliver the result attestation in a asynchronous way permits cost effective implementations of the LTA.

An LTA may deliver immediately a second type response. This occurs for example for a STATUS or LISTIDS function or when an operation is retried and the final result is available.

A client can repeat an operation, since the request or the response might have been lost.

For an ARCHIVE operation, the client MAY provide a unique identification for the request that to be used by the LTA to ensure idempotence of the operation. When retrying an ARCHIVE operation, a client MAY replace the raw data to be archived by a MessageImprint representing the data in order to reduce the payload size.

For the ARCHIVE, DELETE, VERIFY operations, after receipt of the technical acknowledgement (first type response), the client can also use a STATUS operation for the object in order to indirectly determine the outcome of a transaction. Depending on the lower layer bindings, sending a STATUS request or retrying another operation may be the only way to determine the outcome of an archive operation.

2.4. Life cycles of objects

TOC

Using the defined transactions and the operations on objects, two levels of life cycles are defined. One is directly derived from the operation of a single transaction. The other is related to the long-term situation of an object.

2.4.1. Transaction Level Life Cycle

TOC

When a client initiates an archive operation, client and server have some knowledge of the progress of the operation. The following list explains the different states of a transaction seen by both the client and the service, and decribes actions and events changing the state of the transaction.

T0: The client has not initiated an archive operation. The LTA does not know anything about an object.

T1: The client has initiated an archive operation. The LTA may have received the object or not; the client cannot assume that the LTA has received the request. Client may retry the operation after a timeout.

T2: The LTA has received the request and has generated a first type response. The server ensures idempotence of at least the ARCHIVE operation, i.e. on multiple occurrences of an identical request, the LTA sends the same response and transaction identification. The LTA may still lose the state, e.g., in case of a power failure.

T3: The client has received the initial response. The client can retry the initial operation using the transaction identification at the place of the data in order to receive a final answer. In case of negative response, i.e. LTA in state T0, client can fall back to T1.

T4: The LTA has send a definitive answer for an operation and has assumed the responsibility of operation.

T5: The client has received a definitive answer and can consider the operation as terminated.

In case of negative reponses to a transaction, the LTA keeps the result for a certain time in order to provide a reasonable answer to a client that retries an operation.

2.4.2. Long term life cycle.

TOC

We can distinguish the following phases in the lifetimes of an archived object:

L0: The LTA has no knowledge about an object.

L1:

An LTA has received an archive object and is proceeding the request. The LTA may accept or reject the request, or may lose knowledge about it.

L2: An LTA has archived an object. An LTA can accept other operations, i.e., EXPORT, DELETE or VERIFY operations. When receiving an EXPORT or VERIFY operation, some metadata may be updated, e.g., last time of access, last verification operation etc.

L3: After a DELETE operation prior to the initially defined lifetime of the operation, the LTA MAY keep an information about the actual status of the object (e.g. deleted) until the end of the lifetime. If available, the LTA MAY returns other remaining metadata containing for example a new location of the data.

L4: After the lifetime of an object, an object or the reference to is being deleted. At the end of this internal operation, the LTA is in state L0.

When an LTA has archived an object, it keeps a certain number of metadata, which gives information about the current status of the object. Some metadata MAY remain available even after deleting the object. Among the remaining metadata there may be the date of deletion or a reference to where the information can actually be received.
A change from state L2 can also occur when an LTA determines loss of integrity or loss of data.
The LTA MUST maintain a chronological order for references to archived data.

2.5. Roles, Service Types, Policies and Configurations

TOC

The protocol assumes a number of different actors playing different roles. The basic roles are a client and a server. These roles are simply defined by the types of protocol data units, i.e., requests and responses. Several other roles may exist, which are currently not in the scope of the protocol specification. An example of an additional role is a relay or a proxy using both the basic roles of client and server. In general two entities are distinguished, based on different characteristics: an entity that requests its data to be archived or to be acted upon, and an entity that accepts data and assures responsibility of archived data, or acts on the data. Other entities serving as a lower layer transport services, data storage services or security services are out of the scope of protocol definition.
Clients may occur in different roles. Besides users that archive data, there may be relying or controlling entities like a judge who must be

able to get access to it. Or, there are entities like auditors that may access to some data. The protocol distinguishs such roles by the definition of the following service types and service policy information.

The LTA interface implementation MUST enable clients to perform all the service operations.

A client implementation MAY only support a subset of the service types in order to have a small footprint. This is motivated by the fact that different operations are generally invoked by different entities in totally different environments, e.g., a client may only submit data and never verify an evidence record.

The way a particular operation is performed is only defined at the LTA server side implementation and can be influenced by policy information parameters. A client MAY indicate one or more service policy identifiers associated to a service type in order to select different features to be performed by the LTA. The goal of policy identifiers is to keep client configurations simple.

An LTA service may provide additional features, which may be identified by clients, that govern how services are performed. An LTA might offer a series of features based on quality characteristics, e.g. number of timestamps used, refresh period, etc. The protocol specification builds on the assumption that features are clearly identifiable and are included in the protocol elements. Features enable clients to request specific handling by the LTA, such as requesting a premium service that assures prompt and immediate archiving vs. a standard service that handles queues and generates evidence data periodically based on data collections, i.e., one timestamp per a document bundle. Also, services may differ according to data storage characteristics (e.g. client may request full evidence and storage capacity or only evidence creation service), redundancy characteristics (single timestamp versus multiple time stamping), etc. Service characteristics are defined by archive or operation policies.

An LTA may use external services, like validation and evidence creation services. Another service is provision of physical infrastructure or data storage and management systems. Such entities can also be referenced by service policy identifiers.

In general, for each client, in particular those that are archiving, a default or single possible configuration is defined at the server in order to group features and policies into defined sets. A server may operate different configurations and from the protocol standpoint, general configuration is selected by the policy identificator.

As a last mechanism to provide parameters to the archive server, LTA clients MAY use specific configuration parameters in their requests. The definition of such parameters is not in the scope of this protocol. Configuration parameters allow clients to transfer arbitary key/value pairs from the client to the server.

In principle, a single sequence of policy information is sufficient to indicate both the service type and the configuration parameters. A

multi-dimensional approach with configuration and service types rounds up the requirements for LTA and scenarios of archiving processes. This specification defines no particular policy or configuration.

2.6. Identification

TOC

This text defines one ASN.1 module using current ASN.1 syntax. The last compoenet of the module object identifier is a version indication. A value of 0 indicates any draft of this memo.
The ASN.1 Module
Current ASN.1 Module start

```
LTAP {iso(1) identified-organization(3) dod(6)
      internet(1) security(5) mechanisms(5)
      ltans(11) id-mod(0) id-mod-ltap(4) 0}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
```

An equivalent XML schema using XSD notation is also defined. The schema has been generated automatically.
XML Schema Identification

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
  targetNamespace="http://www.setcce.org/schemas/ltap"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
<annotation><documentation xml:lang="en">
   XML Schema for LTAP
</documentation></annotation>
```

2.7. External definitions

TOC

The module export all definitions, and imports several definitions from other modules.
ASN.1 external definitions

```
-- EXPORTS ALL
IMPORTS

  PolicyInformation, GeneralNames
FROM PKIX1Implicit-2009
  {iso(1) identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) id-mod(0)
  id-mod-pkix1-implicit-02(59)}

  CONTENT-TYPE, ContentInfo
FROM CryptographicMessageSyntax2004
  { iso(1) member-body(2) us(840) rsadsi(113549)
  pkcs(1) pkcs-9(9) smime(16) modules(0)
  id-mod-cms-2004-02(41) }
FROM PKCS7
  {iso(1) member-body(2) us(840) rsadsi(113549)
  pkcs(1) pkcs-7(7) modules(0) pkcs-7(1)}
;
```

These are corresponding XML definitions for the imported structures.

XML external definitions

```
<xsd:complexType name="Name">
 <xsd:choice>
   <xsd:element name="rdnSequence" type="RDNSequence"/>
 </xsd:choice>
</xsd:complexType>

<xsd:complexType name="RDNSequence">
 <xsd:sequence minOccurs="0" maxOccurs="unbounded">
  <xsd:element name="RelativeDistinguishedName"
               type="RelativeDistinguishedName"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="RelativeDistinguishedName">
 <xsd:sequence minOccurs="0" maxOccurs="unbounded">
  <xsd:element name="AttributeTypeAndDistinguishedValue"
               type="AttributeTypeAndDistinguishedValue"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AttributeTypeAndDistinguishedValue">
 <xsd:sequence>
  <xsd:element name="type">
   <xsd:simpleType>
    <xsd:restriction base="OBJECT_IDENTIFIER"/>
   </xsd:simpleType>
  </xsd:element>
  <xsd:element name="value">
   <xsd:complexType mixed="true">
    <xsd:choice>
     <xsd:any processContents="lax"/>
    </xsd:choice>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PolicyInformation">
 <xsd:sequence>
  <xsd:element name="policyIdentifier"
               type="CertPolicyId"/>
  <xsd:element name="policyQualifiers" minOccurs="0">
   <xsd:complexType>
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
     <xsd:element name="PolicyQualifierInfo"
                  type="PolicyQualifierInfo"/>
    </xsd:sequence>
   </xsd:complexType>
```

```
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PolicyQualifierInfo">
 <xsd:sequence>
  <xsd:element name="policyQualifierId">
   <xsd:simpleType>
    <xsd:restriction base="OBJECT_IDENTIFIER"/>
   </xsd:simpleType>
  </xsd:element>
  <xsd:element name="qualifier" minOccurs="0">
   <xsd:complexType mixed="true">
    <xsd:choice>
     <xsd:any processContents="lax"/>
    </xsd:choice>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="CertPolicyId">
 <xsd:restriction base="OBJECT_IDENTIFIER"/>
</xsd:simpleType>
```

2.8. Entities

TOC

Entities that participate in protocol exchanges are represented by identifiers and may possess attributes. It is outside the scope of this definition to define an organisation of identifiers and attributes, in particular the way how entity identifiers are related to identifiers used for authentication, or what attributes are associated to data.
As the current LTAP specification assumes end-to-end communication only, there is no distinction between technical roles like 'client, 'server', 'relay', 'proxy' or 'authorized agent'. For LTAP, only client and server roles are defined.
The explicit usage of identifiers and attributes enables decisions to be traceable, i.e., the participating entities can indicate to a certain degree why they want a service or why it has been provided. Furthermore, entity identifiers and attributes MAY be provided by the transport or security layer information. These information can be added to protocol elements as trace attributes.

2.8.1. Entity Identifiers

TOC

Entity identifiers are used in the protocol to indicate the participating entities. A client can indicate one or more identifiers indicating who is making the request or participating in its creation and one or more identifiers indicating who should perform the service. A server can indidate who has provided the service and who is the indented client.
It MUST be ensured in some way that in an actual context of a client/ server network names are scalable and global both in terms of actual community space and time to live of the treated data objects.
Identifiers are labeled in some way, i.e. string representations are typed and can be derived from various external layers. Identifiers SHOULD use an appropriate structure such as ASN.1 definition of GeneralName.

2.8.2. Attributes

TOC

Entities may possess additional attributes like roles, scopes or capabilities. Entities MAY indicate attribute values in protocol exchanges so that they can be used for authentication purposes or billing.
Attributes may be related to attributes of data, for example, an entity may acts as a judge or arbitrator for a particular jurisdiction. The attribute jurisdiction is associated to the entity and to data treated by the service, and thus, can be used for authorisation control.

2.9. Data Model

TOC

The data fields of a LTAP request are as follows:

- request information or status information
- raw data to archive, or references to data or to transactions.
- metadata providing additional information about the data to archive
- authorisation and authentication information of the entities paticipating in the procedure
- other information, required for supporting functions like billing

2.9.1. Data objects

TOC

The data to be archived are arbitrary binary data and, minimally, an associated type that MUST be either available as part of a server configuration policy or explicitly indicated by the client.
Data can be referenced by identifiers. Data identifiers are used to uniquely identify data objects. Data identifiers SHOULD have an additional local structure (e.g., contain a checksum), in order to avoid or detect client copying errors. An additional measure to enhance the redundancy of identifiers is the usage of time values which can be used in combination with data identifiers.
Servers MUST create a server-wide unique identifier for each data object managed by the LTA. The identifier MUST be global during the intended lifetime of an object.
Clients may provide their own data identifiers in requests. Whether the client provided identifiers are unique is outside the scope of the protocol. LTAs treat these identifiers as opaque information.
In order to identify data for the short lifespan of a transaction, artifacts can be used to reference data or transactions.

2.9.2. Collections of objects

TOC

Data grouping can occur for various reasons, i.e. logical, contextual, semantic, operational, etc. Grouping of objects can be performed by a client using metadata present for each object. This document does not specify how client can create groups of objects, collections, hierarchies etc.
Collections of data can be defined explicitly or implicitly. A document amy be implicitelt added to a collection using policy and entity identifiers to request a specific collection strategy (e.g. a collection of data that is processed on a daily basis for a specific user). A collection identifier may be explicitely present as metadata. Grouping can also be implicitely defined by service policies, e.g. per user or on a daily or volume basis. Details are out of scope of this specification.
An LTA MAY understand metadata concerning collections in order to optimize for example evidence creation by building hash trees as defined in [RFC4998] (Gondrom, T., Brandner, R., and U. Pordesch, “Evidence Record Syntax (ERS),” August 2007.) or operational reasons, e.g. for reducing storage costs or to improve performance or scalability.

TOC

2.9.3. MetaData

Meta information is associated with archive data and can be included implicitly, i.e. be a part of a document, or explicitly, i.e. as a document attachment. An LTA does not interprete metadata that may express logical relations among documents in the archive that is submitted selectively using several requests. For LTAs, the client is only in control of selecting and enclosing meta information, which is logically, contextually or for any other reason related to a document. Meta information may occur in various forms and may be an integral part of archive data, e.g. security attributes in form of digital signatures. To process such information, the LTA MUST retrieve enough information on the type and purpose of information enclosed, which may simply be defined with the use of an apropriate archive service policy, e.g. archive service for digitally signed documents.
An LTA may perform specific actions related to meta information processing (and preservation, such as complementary data collection in form of digital certificates). This can also be done by an external service, e.g. DVCS (Adams, C., Sylvester, P., Zolotarev, M., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols," February 2001.) [RFC3029] or SCVP (Freeman, T., Housley, R., Malpani, A., Cooper, D., and W. Polk, "Server-Based Certificate Validation Protocol (SCVP)," December 2007.) [RFC5055].
In some scenarios, a specific set of meta information must be preserved together with archive data, e.g. information identifying the document owner/author, location or time. The LTAP protocol does not define constraints on information type and structure. The LTAP request structure is defined to accept any type of data.

2.9.4. Binding Information

TOC

Clients and servers MAY include additional information in their requests and responses concerning the lower layer binding to a transport like SOAP, HTTP or S/MIME, e.g. end-point addresses. This category may also include things like billing/accounting information, i.e. whatever a business transaction needs but which is not part of metadata, i.e. outside the scope of the archived data.

2.9.5. Evidence Data

TOC

Evidence information demonstrates the integrity and existence of archived data. The LTA accepts data for the single purpose of generating or obtaining evidence information for data submitted by a

client. The evidence information structure is defined in [RFC4998] (Gondrom, T., Brandner, R., and U. Pordesch, “Evidence Record Syntax (ERS),” August 2007.).
In the case where LTA accepts data only for the purpose of generating evidence information (without storage capabilites to avoid, e.g. confidentiality issues), the archivation process is limited in time. When an LTA performs a renewal of evidence, archived data may be required to be available, e.g. when renewing a hash tree. In such scenarios, the LTA requires availability of archived data for hash re-computation. The LTAP protocol does not support function for data re-submission.

---

3. Common Data Types

TOC

A number of data types are common to both requests and responses.

3.1. MessageImprint

TOC

The MessageImprint type is used to store a short representation of data which can be used to link to some data using the result of a one way hash function applied to some data. It is not assumed always identify some data unabiguously (by definition of a hash function), in particlur, collisions may exist now or in the future. Uniqueness is only assumed within a limited set of data objects in time and number for the lifetime of protocol exchanges.
The structure of a MessageImprint is a sequence of an globally defined identification of a hash function and an representation of an octet string encoding a value of the hash function.
ASN.1 MessageImprint

```
MessageImprint ::= SEQUENCE {
  digestAlgorithm DigestMethodType,
  digestValue     DigestValueType
}

DigestMethodType ::= OBJECT IDENTIFIER
DigestValueType ::= OCTET STRING
```

XML MessageImprint

```
<xsd:complexType name="MessageImprint">
 <xsd:sequence>
  <xsd:element name="digestAlgorithm"
               type="DigestMethodType"/>
  <xsd:element name="digestValue"
               type="DigestValueType"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="DigestValueType">
 <xsd:restriction base="OCTET_STRING"/>
</xsd:simpleType>

<xsd:simpleType name="DigestMethodType">
 <xsd:restriction base="OBJECT_IDENTIFIER"/>
</xsd:simpleType>
```

3.2. Artifact

TOC

The Artifact type is used to reference a transaction, or a result of a transaction, returned as a protocol answer in an initial response, to allow retrieval of a response or progress of a transaction later by the initial client or another authorised entity.
ASN.1 Artifact

```
Artifact ::= PrintableString
```

The corresponding XML type is:
XML Artifact

```
   <xsd:simpleType name="Artifact">
 <xsd:restriction base="PrintableString"/>
    </xsd:simpleType>
```

3.3. MetaData

TOC

The Metadata type is a list of open types which can be regarded as key/value pairs giving addtional information concerning:

*the archived data, e.g. type, short description, title, author...

*operational information related to the preservation process, e.g. owner, access rights, dates, ...

*a reference.

The Metadata type is a recursive definition allowing hierarchical metadata structures using a fixed combination of base types for value fields and identifiers. This technique is similar to the one used SNMP for example. There are no open types.
The 'type' identifies in a globally unique way the semantics of the value. The 'oid' choice can be used in a similar way as with the MIBs in SNMP. The 'uri' choce allows to reference metadata defined by URIs.
The semantics 'attribute' choice normally depends on the semantics of a surrounding definition. Global values for 'attribute' may exist, i.e. the choice can be used in the outermost MetaData sequence.
This specification defines one global 'attribute' "datatype". The 'values' item MUST contain one occurence of either a 'stringValue' which indicates a mime-type, or an 'oidValue' or 'uriValue' indicating an FTAM document type.
The semantics are defined by application layers. No attempt is made to recur to some other existing metedata specification, e.g., the Dublin Core. An LTA is free to map metadata.
Since some global metadata are always associated to data objects and necessary for the LTA service, an LTA MUST provide a complete description of all metadata it associates with an archived data object for operational purposes. A client is not required to understand the semantics of metadata.
ASN.1 MetaData

```
MetaData ::= SEQUENCE OF MetaItem

MetaItem ::= SEQUENCE {
  type CHOICE {
    oid OBJECT IDENTIFIER,
    attribute UTF8String,
    uri  IA5String
  } ,
  values SEQUENCE OF value CHOICE {
    oidValue  OBJECT IDENTIFIER,
    stringValue UTF8String,
    uriValue IA5String,
    integerValue INTEGER,
    opaqueValue OCTET STRING,
    composedValue MetaItem
  }
}
```

XML MetaData

```
<xsd:complexType name="MetaData">
 <xsd:sequence minOccurs="0"
               maxOccurs="unbounded">
  <xsd:element name="MetaItem"
               type="MetaItem"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MetaItem">
 <xsd:sequence>
  <xsd:element name="type">
   <xsd:complexType>
    <xsd:choice>
     <xsd:element name="oid"
                  type="OBJECT_IDENTIFIER"/>
     <xsd:element name="attribute"
                  type="UTF8String"/>
     <xsd:element name="uri"
                  type="IA5String"/>
    </xsd:choice>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="values">
   <xsd:complexType>
    <xsd:sequence minOccurs="0"
                  maxOccurs="unbounded">
     <xsd:choice>
      <xsd:element name="oidValue"
                   type="OBJECT_IDENTIFIER"/>
      <xsd:element name="stringValue"
                   type="UTF8String"/>
      <xsd:element name="uriValue"
                   type="IA5String"/>
      <xsd:element name="integerValue"
                   type="INTEGER"/>
      <xsd:element name="opaqueValue"
                   type="OCTET_STRING"/>
      <xsd:element name="composedValue"
                   type="MetaItem"/>
     </xsd:choice>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

3.4. Nonce

TOC

The Nonce type is used to prevent replays of responses for the STATUS operation. It is an opaque OCTET STRING. A client SHOULD NOT encode additional information inside a value.

ASN.1 Nonce

```
Nonce ::= OCTET STRING
```

XML Nonce

```
<xsd:simpleType name="Nonce">
 <xsd:restriction base="OCTET_STRING"/>
</xsd:simpleType>
```

3.5. RawData

TOC

This type is used to encode data to be archived or returned. Three formats for data are possible:

- *binary data
- *textual data
- *structured data

ASN.1 RawData

```
RawData ::= CHOICE {
  binary     OCTET STRING,
  text       UTF8String,
  structured MetaData
}
```

XML RawData

```
<xsd:complexType name="RawData">
 <xsd:choice>
  <xsd:element name="binary" type="OCTET_STRING"/>
  <xsd:element name="text" type="UTF8String"/>
  <xsd:element name="structured" type="MetaData"/>
  </xsd:choice>
</xsd:complexType>
```

3.6. DataOrTransaction

TOC

This choice type is used to identify data by:

*RawData for the data themselves,

*an Artifact identifying a transaction, or

*a reference to the data, e.g. an URI.

ASN.1 DataOrTransaction

```
DataOrTransaction ::= CHOICE {
  data        RawData,
  transaction Artifact,
  dataref     IA5String
}
```

XML DataOrTransaction

```
<xsd:complexType name="DataOrTransaction">
 <xsd:choice>
  <xsd:element name="data" type="RawData"/>
  <xsd:element name="transaction" type="Artifact"/>
  <xsd:element name="dataref" type="IA5String"/>
 </xsd:choice>
</xsd:complexType>
```

3.7. ArchiveData

TOC

This type is used to describe data together with optional metadata and reference information. At least one of the optional elements MUST be provided in order to either provide or identify the data.
A client MUST provide a metadata type to indicate the type of the data, or the type of the data MUST be defined as part of the service policy. For preservation purposes, an LTA must have information on archive data type (e.g., signed or unsigned). If type is not included, it is assumed that data retrieved must be processed as binary string (e.g signatures are not verifed.).
The number of element items allowed in requests or responses depends on the service function.
The optional dataImprint item contains a hash value of data of a dataSpecifier item of corresponding Archivedata element item in (provided for example in a initial ARCHIVE request.) The dataImprintvalue is calculated on choices in the RawData type in the

following way: For the opaque and string choices, the content of the octet string are used and result in the same digest independantly of the actual encoding of the data. For the structured choice the result differs depending on the encoding (XSD or DER).

ASN.1 ArchiveData

```
ArchiveData ::= SEQUENCE OF element SEQUENCE{
  dataSpecifier DataOrTransaction ,
  metaData      MetaData OPTIONAL ,
  dataImprint   [0] MessageImprint OPTIONAL
}
```

XML ArchiveData

```
<xsd:complexType name="ArchiveData">
 <xsd:sequence minOccurs="0" maxOccurs="unbounded">
  <xsd:element name="element">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="data"
                  type="DataOrTransaction"/>
     <xsd:element name="metaData"
                  type="MetaData" minOccurs="0"/>
     <xsd:element name="dataImprint"
                  type="MessageImprint" minOccurs="0"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

3.8. SerialNumber

TOC

A SerialNumber type is an integer type and is used to identify a request and its response. An LTA MAY add an additional verifiable structure, e.g. checksum digits, in order to avoid copying errors in long-term applications with potential media break.

ASN.1 SerialNumber

```
SerialNumber ::= INTEGER
```

XML SerialNumber

```
<xsd:simpleType name="SerialNumber">
 <xsd:restriction base="INTEGER"/>
</xsd:simpleType>
```

3.9. LtapTime

TOC

Clients and servers can add an indication of (its idea of) the time when a request or response was created, search intervals etc. The LtapTime type is used for that purpose. The string content MUST be encoded according to the distinguished encoding rules (DER).
ASN.1 LtapTime

```
LtapTime ::= GeneralizedTime
```

XML LtapTime

```
<xsd:simpleType name="LtapTime">
 <xsd:restriction base="GeneralizedTime"/>
</xsd:simpleType>
```

3.10. Version

TOC

The Version type is used in requests and responses to indicate the protocol version used. This specification is provided for two values:

- *v0 - This version should be used by implementation that want to experiment with draft version of this specification.

- *v1 - this version is used to indicate that the request and response corresponds to this specification.

This memo does not define an extension mechanism at the syntactical level.
ASN.1 Version

```
Version ::= ENUMERATED {
  v0,
  v1
}
```

XML Version

```
<xsd:complexType name="Version">
 <xsd:choice>
  <xsd:element name="v0" type="NULL"/>
  <xsd:element name="v1" type="NULL"/>
 </xsd:choice>
</xsd:complexType>
```

3.11. EntityIdentifiers

TOC

The EntityIdentifiers type are used in the protocol to encode participating entities. A client can indicate one or more identifiers indicating who is making the request or participating in its creation and one or more identifiers indicating who should perform the service. A server can indidate who has provided the service and who is the indented client.
A client MAY also indicate an identifier to return a response in case of an asynchronous operation, e.g. an e-mail address.
It MUST be ensured that in an actual context of a client/server network names are scalable and global both in terms of actual community space and time to live of the treated data objects.
The EntityIdentifiers types is a defined as GeneralNames. The x400Address, ediPartyName MUST NOT be used.
ASN.1 EntityIdentifiers

```
EntityIdentifiers ::= GeneralNames
```

XML EntityIdentifiers

```
<xsd:complexType name="EntityIdentifiers">
 <xsd:sequence minOccurs="0" maxOccurs="unbounded">
  <xsd:choice>
   <xsd:element name="rfc822Name" type="IA5String"/>
   <xsd:element name="dNSName" type="IA5String"/>
   <xsd:element name="directoryName" type="Name"/>
   <xsd:element name="uniformResourceIdentifier"
                type="IA5String"/>
   <xsd:element name="iPAddress" type="OCTET_STRING"/>
   <xsd:element name="registeredID"
                type="OBJECT_IDENTIFIER"/>
  </xsd:choice>
 </xsd:sequence>
</xsd:complexType>
```

3.12. ServiceType

TOC

This ServicetType type is enumeration of the core services defined in this memo and object identifier defined by a service provider. It indicates operations accessible by the protocol.
ASN.1 ServiceType

```
ServiceType ::= CHOICE {
  core ENUMERATED {
    archive,
    delete,
    export,
    status,
    verify,
    listids
  },
  ltapextendedservice OBJECT IDENTIFIER
}
```

XML ServiceType

```
<xsd:complexType name="ServiceType">
 <xsd:choice>
  <xsd:element name="core">
   <xsd:complexType>
    <xsd:choice>
     <xsd:element name="archive" type="NULL"/>
     <xsd:element name="delete" type="NULL"/>
     <xsd:element name="export" type="NULL"/>
     <xsd:element name="status" type="NULL"/>
     <xsd:element name="verify" type="NULL"/>
     <xsd:element name="listids" type="NULL"/>
    </xsd:choice>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="ltapextendedservice"
               type="OBJECT_IDENTIFIER"/>
 </xsd:choice>
</xsd:complexType>
```

3.13. StatusInformation

TOC

The LTA indicates the global status of a transaction using this enumeration type. The semantics of the values is as follows:

waiting
: The response is an initial first type reponse. The request has been technically accepted by the LTA.

granted The response is a second type final response from the LTA.

grantedWithMods The response is a second type final response from the LTA. The operation performed by the LTA but only with some modifications.

error The operation has not been accepted.

more There are more responses available for the LISTIDS function..

In case of modifications or error, the LTA MUST also return details using the following GeneralErrorNotice
ASN.1 StatusInformation

```
StatusInformation ::= ENUMERATED {
  granted,
  grantedWithMods,
  rejection,
  waiting,
  more
}
```

XML StatusInformation

```
<xsd:complexType name="StatusInformation">
 <xsd:choice>
  <xsd:element name="granted" type="NULL"/>
  <xsd:element name="grantedWithMods" type="NULL"/>
  <xsd:element name="rejection" type="NULL"/>
  <xsd:element name="waiting" type="NULL"/>
  <xsd:element name="more" type="NULL"/>
 </xsd:choice>
</xsd:complexType>
```

3.14. RequestInformation

TOC

This type defines a data structure resuming an operation other than the data. The items are filled in by requestor and and may be augmented or modified by the responder. I
ASN.1 RequestInformation

```
RequestInformation ::= SEQUENCE {
  version           Version DEFAULT v1,
  servicePolicyInfo PolicyInformation,
  serviceType       ServiceType,

  requestorID       EntityIdentifier,
  serviceID         EntityIdentifier,
  returnID          EntityIdentifier OPTIONAL,
  serial            SerialNumber OPTIONAL,
  nonce             Nonce OPTIONAL,
  requestTime       LtapTime OPTIONAL,
  startTime         [0] LtapTime OPTIONAL,
  nextTime          [1] LtapTime OPTIONAL,
  bindingInfo       [2] MetaData OPTIONAL
}
```

XML RequestInformation

```
<xsd:complexType name="RequestInformation">
 <xsd:sequence>
  <xsd:element name="version"
               type="Version" minOccurs="0"/>
  <xsd:element name="servicePolicyInfo"
               type="PolicyInformation"/>
  <xsd:element name="serviceType"
               type="ServiceType"/>
  <xsd:element name="requestorID"
               type="EntityIdentifier"/>
  <xsd:element name="serviceID"
               type="EntityIdentifier"/>
  <xsd:element name="returnID"
               type="EntityIdentifier" minOccurs="0"/>
  <xsd:element name="serial"
               type="SerialNumber" minOccurs="0"/>
  <xsd:element name="nonce"
               type="Nonce" minOccurs="0"/>
  <xsd:element name="requestTime"
               type="LtapTime" minOccurs="0"/>
  <xsd:element name="startTime"
               type="LtapTime" minOccurs="0"/>
  <xsd:element name="nextTime"
               type="LtapTime" minOccurs="0"/>
  <xsd:element name="bindingInfo"
               type="MetaData" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
```

The version item indicates the version of the protocol to be used.

The item serviceType indicate the operation to be performed. The value can be one of the enumeration values defined in this memo or an object identifier defining another operation. Additional operations may use metadata for additional parameters.
The serviceID item identifies the LTA service provider(s)
The servicePolicyInfo item defines the service policy.
If a client specifies a nonce item, the server MUST return either the same value, or a value that has the client value as a prefix.
The meaning of the startTime and nextTime items depend on the value of the serviceType item and is described later.
The serial item indicates a serial number of a request has been received, the item MUST NOT be set by the requestor and MUST be set by the responder in order to uniquely identify the request.
The requestTime item indicates the time when the request has been received, the item MAY be set by the requestor in case of a proxy and it SHOULD be set in any response.
The bindingInfo item contains additional information required by the LTA or returned to the client. These metadata are associated with the request and not with the archived data.
The semantics of the items startTime and nextTime depend on the serviceType.

---

4. Top level protocol elements

TOC

On the top level, there are three protocol elements, one is used in requests, and the two other are either describing the successful outcome of an operation or an error notice.

4.1. Request

TOC

This type defined the top level information structure describing a request. It contains a RequestInformation data structure, as well as data or data references. At least one of the data or transactionIdentifier items must be provided. The structure is enveloped in a security or typing envelope LTAPRequest.
ASN.1 Request

```
Request ::= SEQUENCE {
   information           RequestInformation,
   data                  ArchiveData OPTIONAL,
   transactionIdentifier IA5String OPTIONAL
}
```

XML Request

```
<xsd:complexType name="Request">
 <xsd:sequence>
  <xsd:element name="information"
               type="RequestInformation"/>
  <xsd:element name="data"
               type="ArchiveData" minOccurs="0"/>
  <xsd:element name="transactionIdentifier"
               type="IA5String" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
```

4.2. StatusNotice

TOC

A server may return a general error notice indicating an important failure with referencing the request. The element can be created either by the service, e.g., when a request cannot be decoded, but also by a client lower layer, e.g. when a connection cannot be established.
ASN.1 StatusNotice

```
StatusNotice ::= SEQUENCE {
  status           StatusInformation,
  errorInformation UTF8String (SIZE(0..8192)),
  lastValid        LtapTime OPTIONAL,
  transactionIdentifier IA5String OPTIONAL
}
```

XML StatusNotice

```
<xsd:complexType name="StatusNotice">
 <xsd:sequence>
  <xsd:element name="status"
               type="StatusInformation"/>
  <xsd:element name="errorInformation">
   <xsd:complexType mixed="true">
    <xsd:complexContent mixed="true">
     <xsd:extension base="UTF8String"/>
    </xsd:complexContent>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="lastValid"
               type="LtapTime" minOccurs="0"/>
  <xsd:element name="transactionIdentifier"
               type="IA5String" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
```

4.3. OperationResponse

TOC

This structure is returned on a successful or unsuccessful operation of the service. It references the initial request as well as the data that had been submitted.

ASN.1 OperationResponse

```
OperationResponse ::= SEQUENCE {
  information RequestInformation,
  status      StatusNotice,
  data        ArchiveData
}
```

XML OperationResponse

```
<xsd:complexType name="OperationResponse">
 <xsd:sequence>
  <xsd:element name="information"
               type="RequestInformation"/>
  <xsd:element name="status" type="StatusNotice"/>
  <xsd:element name="data" type="ArchiveData"/>
 </xsd:sequence>
</xsd:complexType>
```

4.4. Response

TOC

This type is the top level information structure returned on a successful or unsuccessful operation of the service. It is included in security or typing envelope LTAPResponse.
ASN.1 Response

```
Response ::= CHOICE {
  operationResponse OperationResponse,
  errorNotice   [0] StatusNotice
}
```

XML Response

```
<xsd:complexType name="Response">
 <xsd:choice>
  <xsd:element name="operationResponse"
               type="OperationResponse"/>
  <xsd:element name="errorNotice"
               type="StatusNotice"/>
 </xsd:choice>
</xsd:complexType>
```

---

5. Service Operations

TOC

This section describes in detail the different operations that a client can initiate with a request and their outcomes. All operations share the same data types for the input and output but the choices are filled in differently.
For all operations, the archive service may react in the following ways:

Error The request is not understood or cannot be transmitted, an ErrorNotice is returned.

Acceptance The request is accepted, an OperationResponse indicating that the transaction is 'waiting'. The item nextTime in the requestInformation SHOULD be set by the responder in case when

> the client needs to poll for the final response to indicate to the client not poll before that date.
>
> **Rejection** The request is rejected. an OperationResponse describing the error is returned.
>
> **Result** This a final response, an OperationResponse containing the result of the transaction is returned.

When the transport layer is asynchronous, the protocol is the following: The client MUST fill the item 'returnID' in the requestInformation. A server MAY respond either with the final result or with two messages indicating acceptance and result. The client MAY retry the operation (slightly modified) after that date. Since any operation can be lost, a client has to set an appropriate value for initial retry timeout.
When the transport layer is synchronous, e.g. if HTTP is used, the server immediately returns either an error or acceptance, and the client MUST retry the operation (slightly modified) in order to get the final result.
A server may offer a mixed environment where the initial response is obtained in a synchronous way, and the final response can be transferred in an asynchronous way. In this case, the client MAY be required to set a returnID.

5.1. ARCHIVE operation

TOC

The major operation of the archive service is the ARCHIVE operation. A client prepares the data and associated metadata, and transfers the request to the archive service. The client builds a Request with an information item including service policy interpreting service characteristics and service configuration parameters. Data to be archived and metadata are enclosed.

- the service type is set to "archive"
- If this is an initial request, the data are filled in with one element and rawData.
- The messageImprint item MAY be set in order to perform an integrity check on the rawData. The LTA calculated a messageImprint and returns the calcumated value in the corresponding item of the response. If a messageImprint is present in the request, the server compares it to the calculated value, and rejects the request in case the two value differ.

*If this is a retry of a previous operation, the client has several option depending on the outcome of the previous operation and available information. If no response has been received by the client, the client MAY either repeat the operation as is, or send remove the content octets of the selected choice (replacing it by a zero length choice) and send a messageImprint.

*If available from an initial response, the client MUST provide an artifact, if available from an response.

The server SHOULD use an artifact when the StatusInformation is 'waiting', in order to simplify its processing when the client retries the operation. When an operation is retried, the client SHOULD use the returned artifact instead of the data.
The final response contains a data reference to be usable in other operations concerning the same data object.
When a client archives may objects in parallel operations, it may be unreasonable to poll for each outstanding result individually. An LTA MAY support a LISTIDS function returning artifacts for finished archive operations.

5.2. EXPORT operation

TOC

This operation allows a client to retrieve data.

*Data identification must be a data reference and/or a message imprint.

*The service type is set to "export".

In the final result, the LTA returns data and metadata of the object.

5.3. DELETE operation

TOC

This operation allows a client to delete data or request data shredding. After a successful operation, the the server does not maintain any status information about the object.

*Data identification must include data itself or data reference or message imprint.

*The metadata MAY be set to replace the existing metadata of the object.

*The service type is set to "delete".

The LTA MAY either return a result with updated metadata or nothing. If the client retries a delete operation, it may happen that the LTA has already deleted all traces of the operation. In this case, the server always pretends having deleted the referenced data. The client cannot distinguish whether the data have ever existed.

5.4. VERIFY operation

TOC

This operation allows a client to verify the authenticity of information stored in the archive. Depending on the actual status of the object and on the policy of the LTA, the LTA initiates an internal procedure to determine the validity of the data. An LTA may perform the similar steps as for the initial archiving operation. The LTA MAY choose not to perform the operation if the actual status is sufficiently recent. In this case, the operation is identical to STATUS operation.

*Data identification must be a data reference and/or a message imprint.

*The service type is set to "verify".

The LTA returns updated metadata of the object.

5.5. STATUS operation

TOC

A client can request the status of a data object.
Client builds a Request with request information including service policy interpreting service characteristics and service configuration parameters.

*The service type is set to "status".

*Data identification must be a data reference and/or a message imprint..

*If a nonce is present in a request, a server MUST respond with a response containing the value provided in the request. This does not mean that the server must determine the actual status in a particular backend operation.

In the final response, the LTA returns the current status and the metadata for the object.

5.6. LISTIDS operation

TOC

A client can request a list of reference of objects archived. The client builds a Request with request information including service policy interpreting service characteristics and service configuration parameters.

*The service type is set to "LISTIDS".

*Data identification MUST be a data reference, an artifact and/or a message imprint. If present, the LTA returns references to objects starting with this reference in the chronological order defined by the LTA. The reference or artifact MUST be known to the LTA.

*The startDate and endDate MAY be set to indidate a range. The exact definition of when an object belongs to that range is defined by the LTA.

The LTA returns a list of references as a sequence of DataOrTransaction items. The artifact and dataReference choices are allowed. If the request contains an artifact, the LTA MUST return the artifacts that correspond to terminated transactions.
The number of references returned is defined by the LTA and may not be the complete set. If the LTA has more data to provide, it sets the StatusInformation to 'more'. The client has to repeat the operation using the last returned reference as data identification. A client can distinguish this case from the initial acknowledge which has the same value for StatusInformation. For the Acceptance response no references are returned. There may be a final response with no references incase when no data exist for the specified criteria.

---

6. Presentation and Bindings

TOC

In the previous chapters we have presented all basic data types as well as XSD schema as in with ASN.1. This is done in order to allow implentations work on both data syntaxes and to be able to present and transform messages in a defined way.
There is no mandatory transport mechanism in this document. All mechanisms are optional. Two examples of transport protocols are given that allow online exchange of request and a response, and asynchronous

communication between a client and an LTA. An LTA MAY use a combination of protocols, for example in order to return additional responses. This memo defines bindings for the transfer of requests and/or responses, one using HTTP and another using e-mail.

6.1. Common parameters and encoding requirements

TOC

This memo defines two principalways how requests and responses are encoded, either using a restricted BER encoding or XML. An LTA MUST provide at least one of them. Furthermore, we define optional enveloping protection mechanisms which depend on the encoding. CMS protection of signedData and envelopedData can be used independantly of the encoding of the request or response. XML-DSIG and XML-ENC can only be used for XML encoded requests and responses.
For requests encoded in XML either based on XER (or the equivalent XSD), the associated MIME type is application/ltap-request+xml. For request not encoded in XML, the associated MIME type is application/ltap-request.
Similarly, for responses have an associated MIME types application/ltap-response and application/ltap-response+xml.
When request and responses are exchanged using an XML encoding, the XSD top level elements LTAPRequest or LTAPResponse are used, and not an XER version of ContentInfo.
When a XML Signature is used, an enveloping signature for a Request or Response or enc:EncryptedData MUST be used
EncryptedData MUST decrypt to a Request or Response or ds:Signature
When the request is presented with the application/ltap-request type, the client MAY encode it using BER with strings nested at most one level. Similarily, a response presented with the application/ltap-response type may have BER with strings nested at most one level.
The Request and Response items can be encapsulated inside CMS signedData and/or CMS envelopedData, or, If not protected, the Request is encapsulated in a ContentInfo using id-ct-LTAPRequest as identification, and the Response is encapsulated in a ContentInfo structure using id-ct-LTAPResponse for identification.
If the Request and Response which are not encoded in XML are encapsulated inside SignedData and/or EnvelopedData, the contenttype of the innermost encapsulatedContent is set to using id-ct-LTAPRequest or id-ct-LTAPResponse respectively.
Any of the four MIME parts can be encapsulated inside CMS using an id-data content-type.
LTANS has its own object identifier tree, the content-types are defined there. The owner of the S/MIME arc doesn't like to register them in the S/MIME arc.
CMS ContentTypes identifiers

```
id-ltans-ct OBJECT IDENTIFIER ::= {
  iso(1) identified-organization(3)
  dod(6) internet(1) security(5) mechanisms(5)
  ltans(11) 1 }

id-ct-LTAPRequest  OBJECT IDENTIFIER ::=
  { id-ltans-ct 4 }
id-ct-LTAPResponse  OBJECT IDENTIFIER ::=
  { id-ltans-ct 5 }
```

In current ASN.1 we have the following ContentType definitions.
CMS ContentTypes

```
ltap-Request  CONTENT-TYPE ::= {Request
  IDENTIFIED BY id-ct-LTAPRequest }
ltap-Response CONTENT-TYPE ::= {Response
  IDENTIFIED BY id-ct-LTAPResponse }

LTAPRequest ::= ContentInfo
LTAPResponse ::= ContentInfo
```

XML top level definitions

```
<xsd:element name="Request" type="Request"/>
<xsd:element name="Response" type="Response"/>
<xsd:element name="LTAPRequest" type="LTAPRequest"/>
<xsd:element name="LTAPResponse" type="LTAPResponse"/>

<xsd:complexType name="LTAPResponse">
 <xsd:choice>
  <xsd:element name="response" type="Response"/>
  <xsd:element name="signedResponse" type="ds:Signature"/>
  <xsd:element name="encryptedResponse" type="enc:EncryptedData"/>
 </xsd:choice>
</xsd:complexType>

<xsd:complexType name="LTAPRequest">
 <xsd:choice>
  <xsd:element name="request" type="Request"/>
   <xsd:element name="signedRequest" type="xd:Signature"/>
   <xsd:element name="encryptedRequest" type="enc:EncryptedData"/>
 </xsd:choice>
</xsd:complexType>
```

TOC

6.2. e-mail bindings

In a request, the last element of item requestorID corresponds to the From header, the last element of the item serviceID to the To header and the last element of the returnID to reply-to header. The e-mail header is not used by the LTA, but rather the items in the requestInformation. When a server acts as a relay, it MAY add appropriate values to these items.
For the response, an LTA normally sets the From and To header fields to either the last items of serviceID and returnID (or requestorID). In case of a relaying LTA, when the LTA receives a response from another LTA, it first determines its own identity in requestInformation, sets the From: header to its own identity and the To: header to the identity of its client.

6.3. HTTP Bindings

TOC

Servers MUST understand HTTP 1.1 requests at least for the ARCHIVE, EXPORT and LISTIDS functions allowing chunked input of a POST request and 'Continue' responses. A server SHOULD understand a Content-Encoding value of gzip. In case of a HTTP 1.0 request and response, a positive value Content-Length indicating the total size of the data MUST be used. A client MUST send a Host header in the request.
The Request URI may be used to indicate a particular service endpoint. When using HTTPS, TLS MUST be supported by clients and servers. Clients SHOULD send a TLS servername extension in the ClientHello.
The Content-Type header MUST be set to "application/ltap-request" or "application/ltap-request+xml". The LTAPrequest message MUST be sent in the body of the HTTP Request.
The Content-Type header MUST be set to "application/ltap-response" or "application/ltap-response+xml". The LTAP response message MUST be sent in the body of the HTTP Response.
The HTTP status code MUST be set to 200 if a LTAP response message is returned. Otherwise, the status code can be set to 3xx to indicate a redirection, 4xx to indicate a low-level client error (such as a malformed request), or 5xx to indicate a low-level server error.
Clients SHOULD react automatically to redirections.
When using HTTPS, TLS 1.0 MUST be supported. SSL 3.0 MAY be supported by servers. Future versions of TLS MAY be supported. Clients SHOULD send a TLS servername extension in the ClientHello.
RSA ciphersuites MUST be supported. Diffie-Hellman and DSS ciphersuites MAY be supported. TripleDES ciphersuites MUST be supported. AES ciphersuites SHOULD be supported.

TOC

6.4. Security

A request and a response MAY be encapsulated in an [RFC3852] (Housley, R., “Cryptographic Message Syntax (CMS),” July 2004.) signedData or envelopedData where the content type indicated in the eContentType of the encapContentInfo is one of the LTAP content types and the eContent of the encapContentInfo, carried as an octet string containing an encoded request or response structure.

When using a SignedData structure for authentication, LTAP requests and responses MAY contain one or more SignerInfo structures, each of which may contain countersignature attributes depending on operational environments. Relaying LTAs MAY add additional signatures or a countersignature attributes or remove the encapsulation and create a new one depending on the requirements of the next LTA.

For the XML encoded structures, alternatively, security mechanisme from [W3C.xmldsig-core] (Eastlake, D., Reagle , J., and D. Solo, “XML-Signature Syntax and Processing,” October 2000.) and [W3C.xmlenc-core] (Eastlake, D. and J. Reagle , “XML Encryption Syntax and Processing,” August 2002.) may be used. An LTA MAY impose restrictions on the usage of these features.

Clients and relays MUST ensure authenticity of a server when submitting data. In order to do so, they MAY add another encapsulation from [RFC3852] (Housley, R., “Cryptographic Message Syntax (CMS),” July 2004.) that provides for confidentiality, and/or MAY use a secure transport layer, e.g., TLS to perform server authentication and to ensure confidentiality of the transport.

Responses are generally protected in similar way by using a SignedData encapsulation with one or more SignerInfos, and CounterSignatures, depending on the number of participating servers. The number of signatures is not related to the number of participating servers but rather to the number of entities that may be used to authenticate a response or part of it.

In some circumstances, a client/server communication may be secured only by lower layer transport mechanism, e.g. SSL/TLS.

A client MUST NOT trust a response that cannot be authenticated.

Archive clients and servers MUST always create requests and responses that can be authenticated with the explicit exception of a global error status, which may be returned as a non-signed response.

In order to be able to associate a possible error response with a request, the requester SHOULD use the item 'transactionIdentifier'. The requester SHOULD NOT make any assumption about the usage of message header fields by the responding service, in particular the usage of fields like Subject, Message-ID or References.

TOC

7. Credits

This document has been created using XML2RFC ([RFC2629] (Rose, M., “Writing I-Ds and RFCs using XML,” June 1999.)).
The ASN.1 and XSD modules have been automatically collected from the definitions in the other paragraphs using a small tool written by Peter Sylvester. This tool can also extract the modules from the xml source.
The XSD schema has been generated automatically using the asn1xsd tool from OSS Nokalva. It has been manually
The ASN.1 has been validated using the asn1c compiler from OSS Nokalva.

---

8. Security Considerations

TOC

This section discusses addition security considerations of the framework.
When designing an LTA service, the following considerations have been identified that have an impact upon the validity or "trust" in the ltans server responses.
An LTA is assumed to operate with best effort. Nevertheless, an operation can fail or get totally lost. A client SHOULD be able to recover from lost requests, i.e., avoid deleting data until an attestation has been received.
It is possible for an LTA to report loss of integrity for archived data, or simply non-existence of data which is equivalent to loss of data. Depending on the value of the data, appropriate measures to address these catastrophic scenarios need to be provided outside the core service, e.g., by using redundant copies managed either by a client of internally of a broker type service.
The validity of data should be checked by periodic execution of VERIFY operations intended to ensure data with demonstratable integrity is available throughout the lifetime of an archived data object. The rate of refresh will be driven by a number of factors, some of which have a direct impact of demonstration of integrity. For example, the confidence in the strength of cryptographic algorithms or the quality of storage devices are factors determining the verification intervals. Depending on the lifetime and the quality of data, relying on cryptographic protection of data object may not be a sufficient means to determine authenticity in time, other means may be required, e.g. physical protection of data storage material.
It is imperative that keys used to sign responses are guarded with proper security and controls in order to minimize the possibility of compromise. Nevertheless, in case the private key does become compromised, an audit trail of all the response generated by the service SHOULD be kept as a means to help discriminate between genuine and false responses. An LTA MAY provide for a service to validate

responses created by this service or another one solely based on the audit trail.
As already indicated, when confidentiality and server authentication is required, requests and responses MAY be protected using appropriate mechanisms (e.g., CMS encapsulation [RFC3852] (Housley, R., "Cryptographic Message Syntax (CMS)," July 2004.) or TLS [RFC5246] (Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," August 2008.)).
Server authentication is highly recommended for all service which transfer data to a server.
Client identification and authentication MAY use services defined by TLS ([RFC5246] (Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," August 2008.)) instead of, or in addition to, using a document or message protection format, e.g. CMS.
It is possible for an LTA to report loss of integrity for archived data, or simply non-existence of data which is equivalent to loss of data. Depending on the value of the data, appropriate measures to address these catastrophic scenarios need to be provided outside the core service, e.g., by using redundant copies managed either by a client of internally of a broker type service.

---

9. IPR Patent Information

TOC

The material presented in this document was initially drafted in 2005. The following United States Patents related to data validation and certification services, listed in chronological order, are known by the authors to exist at this time. This may not be an exhaustive list. Other patents may exist or be issued at any time. Implementers of this protocol and applications using the protocol SHOULD perform their own patent search and determine whether or not any encumberences exist on their implementation. The list is intitially taken from [RFC3029] (Adams, C., Sylvester, P., Zolotarev, M., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols," February 2001.).
4,309,569 Method of Providing Digital Signatures
(issued) January 5, 1982
(inventor) Ralph C. Merkle
(assignee) The Board of Trustees of the Leland Stanford Junior University

5,001,752 Public/Key Date-Time Notary Facility
(issued) March 19, 1991
(inventor) Addison M. Fischer

5,022,080 Electronic Notary
(issued) June 4, 1991

(inventors) Robert T. Durst, Kevin D. Hunter

5,136,643 Public/Key Date-Time Notary Facility
(issued) August 4, 1992
(inventor) Addison M. Fischer

(Note: This is a continuation of patent # 5,001,752.)
5,136,646 Digital Document Time-Stamping with Catenate Certificate
(issued) August 4, 1992

(inventors) Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Bell Communications Research, Inc.

5,136,647 Method for Secure Time-Stamping of Digital Documents
(issued) August 4, 1992
(inventors) Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Bell Communications Research, Inc.

5,373,561 Method of Extending the Validity of a Cryptographic Certificate
(issued) December 13, 1994
(inventors) Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Bell Communications Research, Inc.,

5,422,95 Personal Date/Time Notary Device
(issued) June 6, 1995
(inventor) Addison M. Fischer

5,781,629 Digital Document Authentication System
(issued) July 14, 1998
(inventor) Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Surety Technologies, Inc.

---

10. IANA considerations

TOC

LTAP request and response messages are identified using Object Identifiers (OIDs), which are defined in an arc delegated by IANA to the LTANS Working Group. This document also includes four MIME type registrations in Section 6.3 (HTTP Bindings). No further action by IANA is necessary for this document.

---

11. References

TOC

11.1. Normative references

TOC

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[RFC3852]	Housley, R., "Cryptographic Message Syntax (CMS)," RFC 3852, July 2004 (TXT).
[RFC4998]	Gondrom, T., Brandner, R., and U. Pordesch, "Evidence Record Syntax (ERS)," RFC 4998, August 2007 (TXT).
[RFC5246]	Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, August 2008 (TXT).

11.2. Informative references

TOC

[RFC2629]	Rose, M., "Writing I-Ds and RFCs using XML," RFC 2629, June 1999 (TXT, HTML, XML).
[RFC3029]	Adams, C., Sylvester, P., Zolotarev, M., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols," RFC 3029, February 2001 (TXT).
[RFC4810]	Wallace, C., Pordesch, U., and R. Brandner, "Long-Term Archive Service Requirements," RFC 4810, March 2007 (TXT).
[RFC5055]	Freeman, T., Housley, R., Malpani, A., Cooper, D., and W. Polk, "Server-Based Certificate Validation Protocol (SCVP)," RFC 5055, December 2007 (TXT).
[W3C.xmldsig-core]	Eastlake, D., Reagle , J., and D. Solo, "XML-Signature Syntax and Processing," W3C Recommendation xmldsig-core, October 2000.
[W3C.xmlenc-core]	Eastlake, D. and J. Reagle , "XML Encryption Syntax and Processing," W3C Candidate Recommendation xmlenc-core, August 2002.

Appendix A. ASN.1 module

TOC

The following ASN.1 module has been checked using the asn1c tool.

```
LTAP {iso(1) identified-organization(3) dod(6)
      internet(1) security(5) mechanisms(5)
      ltans(11) id-mod(0) id-mod-ltap(4) 0}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN


-- EXPORTS ALL
IMPORTS

  PolicyInformation, GeneralNames
FROM PKIX1Implicit-2009
  {iso(1) identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) id-mod(0)
  id-mod-pkix1-implicit-02(59)}

  CONTENT-TYPE, ContentInfo
FROM CryptographicMessageSyntax2004
  { iso(1) member-body(2) us(840) rsadsi(113549)
  pkcs(1) pkcs-9(9) smime(16) modules(0)
  id-mod-cms-2004-02(41) }
FROM PKCS7
  {iso(1) member-body(2) us(840) rsadsi(113549)
  pkcs(1) pkcs-7(7) modules(0) pkcs-7(1)}
;

MessageImprint ::= SEQUENCE {
  digestAlgorithm DigestMethodType,
  digestValue     DigestValueType
}

DigestMethodType ::= OBJECT IDENTIFIER
DigestValueType ::= OCTET STRING


Artifact ::= PrintableString

MetaData ::= SEQUENCE OF MetaItem

MetaItem ::= SEQUENCE {
  type CHOICE {
    oid OBJECT IDENTIFIER,
    attribute UTF8String,
    uri  IA5String
  } ,
  values SEQUENCE OF value CHOICE {
    oidValue  OBJECT IDENTIFIER,
    stringValue UTF8String,
    uriValue IA5String,
```

```
    integerValue INTEGER,
    opaqueValue OCTET STRING,
    composedValue MetaItem
  }
}

Nonce ::= OCTET STRING

RawData ::= CHOICE {
  binary     OCTET STRING,
  text       UTF8String,
  structured MetaData
}

DataOrTransaction ::= CHOICE {
  data        RawData,
  transaction Artifact,
  dataref     IA5String
}

ArchiveData ::= SEQUENCE OF element SEQUENCE{
  dataSpecifier DataOrTransaction ,
  metaData      MetaData OPTIONAL ,
  dataImprint   [0] MessageImprint OPTIONAL
}


SerialNumber ::= INTEGER

LtapTime ::= GeneralizedTime

Version ::= ENUMERATED {
  v0,
  v1
}

EntityIdentifiers ::= GeneralNames

ServiceType ::= CHOICE {
  core ENUMERATED {
    archive,
    delete,
    export,
    status,
    verify,
    listids
  },
  ltapextendedservice OBJECT IDENTIFIER
}
```

```
StatusInformation ::= ENUMERATED {
  granted,
  grantedWithMods,
  rejection,
  waiting,
  more
}

RequestInformation ::= SEQUENCE {
  version           Version DEFAULT v1,
  servicePolicyInfo PolicyInformation,
  serviceType       ServiceType,

  requestorID       EntityIdentifier,
  serviceID         EntityIdentifier,
  returnID          EntityIdentifier OPTIONAL,
  serial            SerialNumber OPTIONAL,
  nonce             Nonce OPTIONAL,
  requestTime       LtapTime OPTIONAL,
  startTime         [0] LtapTime OPTIONAL,
  nextTime          [1] LtapTime OPTIONAL,
  bindingInfo       [2] MetaData OPTIONAL
}

Request ::= SEQUENCE {
   information           RequestInformation,
   data                  ArchiveData OPTIONAL,
   transactionIdentifier IA5String OPTIONAL
}


StatusNotice ::= SEQUENCE {
  status           StatusInformation,
  errorInformation UTF8String (SIZE(0..8192)),
  lastValid        LtapTime OPTIONAL,
  transactionIdentifier IA5String OPTIONAL
}

OperationResponse ::= SEQUENCE {
  information RequestInformation,
  status      StatusNotice,
  data        ArchiveData
}

Response ::= CHOICE {
  operationResponse OperationResponse,
  errorNotice   [0] StatusNotice
}

id-ltans-ct OBJECT IDENTIFIER ::= {
```

```
  iso(1) identified-organization(3)
  dod(6) internet(1) security(5) mechanisms(5)
  ltans(11) 1 }

id-ct-LTAPRequest  OBJECT IDENTIFIER ::=
  { id-ltans-ct 4 }
id-ct-LTAPResponse  OBJECT IDENTIFIER ::=
  { id-ltans-ct 5 }


ltap-Request  CONTENT-TYPE ::= {Request
  IDENTIFIED BY id-ct-LTAPRequest }
ltap-Response CONTENT-TYPE ::= {Response
  IDENTIFIED BY id-ct-LTAPResponse }

LTAPRequest ::= ContentInfo
LTAPResponse ::= ContentInfo

END
```

TOC

Appendix B. XML schema for LTAP

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
  targetNamespace="http://www.setcce.org/schemas/ltap"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
<annotation><documentation xml:lang="en">
   XML Schema for LTAP
</documentation></annotation>


<xsd:complexType name="Name">
 <xsd:choice>
   <xsd:element name="rdnSequence" type="RDNSequence"/>
 </xsd:choice>
</xsd:complexType>

<xsd:complexType name="RDNSequence">
 <xsd:sequence minOccurs="0" maxOccurs="unbounded">
  <xsd:element name="RelativeDistinguishedName"
               type="RelativeDistinguishedName"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="RelativeDistinguishedName">
 <xsd:sequence minOccurs="0" maxOccurs="unbounded">
  <xsd:element name="AttributeTypeAndDistinguishedValue"
               type="AttributeTypeAndDistinguishedValue"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AttributeTypeAndDistinguishedValue">
 <xsd:sequence>
  <xsd:element name="type">
   <xsd:simpleType>
    <xsd:restriction base="OBJECT_IDENTIFIER"/>
   </xsd:simpleType>
  </xsd:element>
  <xsd:element name="value">
   <xsd:complexType mixed="true">
    <xsd:choice>
     <xsd:any processContents="lax"/>
    </xsd:choice>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="PolicyInformation">
 <xsd:sequence>
  <xsd:element name="policyIdentifier"
               type="CertPolicyId"/>
  <xsd:element name="policyQualifiers" minOccurs="0">
   <xsd:complexType>
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
     <xsd:element name="PolicyQualifierInfo"
                  type="PolicyQualifierInfo"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PolicyQualifierInfo">
 <xsd:sequence>
  <xsd:element name="policyQualifierId">
   <xsd:simpleType>
    <xsd:restriction base="OBJECT_IDENTIFIER"/>
   </xsd:simpleType>
  </xsd:element>
  <xsd:element name="qualifier" minOccurs="0">
   <xsd:complexType mixed="true">
    <xsd:choice>
     <xsd:any processContents="lax"/>
    </xsd:choice>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="CertPolicyId">
 <xsd:restriction base="OBJECT_IDENTIFIER"/>
</xsd:simpleType>


<xsd:complexType name="MessageImprint">
 <xsd:sequence>
  <xsd:element name="digestAlgorithm"
               type="DigestMethodType"/>
  <xsd:element name="digestValue"
               type="DigestValueType"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="DigestValueType">
 <xsd:restriction base="OCTET_STRING"/>
</xsd:simpleType>
```

```
<xsd:simpleType name="DigestMethodType">
 <xsd:restriction base="OBJECT_IDENTIFIER"/>
</xsd:simpleType>

   <xsd:simpleType name="Artifact">
 <xsd:restriction base="PrintableString"/>
    </xsd:simpleType>

<xsd:complexType name="MetaData">
 <xsd:sequence minOccurs="0"
               maxOccurs="unbounded">
  <xsd:element name="MetaItem"
               type="MetaItem"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MetaItem">
 <xsd:sequence>
  <xsd:element name="type">
   <xsd:complexType>
    <xsd:choice>
     <xsd:element name="oid"
                  type="OBJECT_IDENTIFIER"/>
     <xsd:element name="attribute"
                  type="UTF8String"/>
     <xsd:element name="uri"
                  type="IA5String"/>
    </xsd:choice>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="values">
   <xsd:complexType>
    <xsd:sequence minOccurs="0"
                  maxOccurs="unbounded">
     <xsd:choice>
      <xsd:element name="oidValue"
                   type="OBJECT_IDENTIFIER"/>
      <xsd:element name="stringValue"
                   type="UTF8String"/>
      <xsd:element name="uriValue"
                   type="IA5String"/>
      <xsd:element name="integerValue"
                   type="INTEGER"/>
      <xsd:element name="opaqueValue"
                   type="OCTET_STRING"/>
      <xsd:element name="composedValue"
                   type="MetaItem"/>
     </xsd:choice>
```

```
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="Nonce">
 <xsd:restriction base="OCTET_STRING"/>
</xsd:simpleType>

<xsd:complexType name="RawData">
 <xsd:choice>
  <xsd:element name="binary" type="OCTET_STRING"/>
  <xsd:element name="text" type="UTF8String"/>
  <xsd:element name="structured" type="MetaData"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="DataOrTransaction">
 <xsd:choice>
  <xsd:element name="data" type="RawData"/>
  <xsd:element name="transaction" type="Artifact"/>
  <xsd:element name="dataref" type="IA5String"/>
 </xsd:choice>
</xsd:complexType>

<xsd:complexType name="ArchiveData">
 <xsd:sequence minOccurs="0" maxOccurs="unbounded">
  <xsd:element name="element">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="data"
                  type="DataOrTransaction"/>
     <xsd:element name="metaData"
                  type="MetaData" minOccurs="0"/>
     <xsd:element name="dataImprint"
                  type="MessageImprint" minOccurs="0"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="SerialNumber">
 <xsd:restriction base="INTEGER"/>
</xsd:simpleType>

<xsd:simpleType name="LtapTime">
 <xsd:restriction base="GeneralizedTime"/>
```

```
</xsd:simpleType>

<xsd:complexType name="Version">
 <xsd:choice>
  <xsd:element name="v0" type="NULL"/>
  <xsd:element name="v1" type="NULL"/>
 </xsd:choice>
</xsd:complexType>

<xsd:complexType name="EntityIdentifiers">
 <xsd:sequence minOccurs="0" maxOccurs="unbounded">
  <xsd:choice>
   <xsd:element name="rfc822Name" type="IA5String"/>
   <xsd:element name="dNSName" type="IA5String"/>
   <xsd:element name="directoryName" type="Name"/>
   <xsd:element name="uniformResourceIdentifier"
                type="IA5String"/>
   <xsd:element name="iPAddress" type="OCTET_STRING"/>
   <xsd:element name="registeredID"
                type="OBJECT_IDENTIFIER"/>
   </xsd:choice>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ServiceType">
 <xsd:choice>
  <xsd:element name="core">
   <xsd:complexType>
    <xsd:choice>
     <xsd:element name="archive" type="NULL"/>
     <xsd:element name="delete" type="NULL"/>
     <xsd:element name="export" type="NULL"/>
     <xsd:element name="status" type="NULL"/>
     <xsd:element name="verify" type="NULL"/>
     <xsd:element name="listids" type="NULL"/>
    </xsd:choice>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="ltapextendedservice"
               type="OBJECT_IDENTIFIER"/>
 </xsd:choice>
</xsd:complexType>

<xsd:complexType name="StatusInformation">
 <xsd:choice>
  <xsd:element name="granted" type="NULL"/>
  <xsd:element name="grantedWithMods" type="NULL"/>
  <xsd:element name="rejection" type="NULL"/>
  <xsd:element name="waiting" type="NULL"/>
```

```
  <xsd:element name="more" type="NULL"/>
 </xsd:choice>
</xsd:complexType>

<xsd:complexType name="RequestInformation">
 <xsd:sequence>
  <xsd:element name="version"
               type="Version" minOccurs="0"/>
  <xsd:element name="servicePolicyInfo"
               type="PolicyInformation"/>
  <xsd:element name="serviceType"
               type="ServiceType"/>
  <xsd:element name="requestorID"
               type="EntityIdentifier"/>
  <xsd:element name="serviceID"
               type="EntityIdentifier"/>
  <xsd:element name="returnID"
               type="EntityIdentifier" minOccurs="0"/>
  <xsd:element name="serial"
               type="SerialNumber" minOccurs="0"/>
  <xsd:element name="nonce"
               type="Nonce" minOccurs="0"/>
  <xsd:element name="requestTime"
               type="LtapTime" minOccurs="0"/>
  <xsd:element name="startTime"
               type="LtapTime" minOccurs="0"/>
  <xsd:element name="nextTime"
               type="LtapTime" minOccurs="0"/>
  <xsd:element name="bindingInfo"
               type="MetaData" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Request">
 <xsd:sequence>
  <xsd:element name="information"
               type="RequestInformation"/>
  <xsd:element name="data"
               type="ArchiveData" minOccurs="0"/>
  <xsd:element name="transactionIdentifier"
               type="IA5String" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>


<xsd:complexType name="StatusNotice">
 <xsd:sequence>
  <xsd:element name="status"
               type="StatusInformation"/>
```

```
  <xsd:element name="errorInformation">
   <xsd:complexType mixed="true">
    <xsd:complexContent mixed="true">
     <xsd:extension base="UTF8String"/>
    </xsd:complexContent>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="lastValid"
               type="LtapTime" minOccurs="0"/>
  <xsd:element name="transactionIdentifier"
               type="IA5String" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="OperationResponse">
 <xsd:sequence>
  <xsd:element name="information"
               type="RequestInformation"/>
  <xsd:element name="status" type="StatusNotice"/>
  <xsd:element name="data" type="ArchiveData"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Response">
 <xsd:choice>
  <xsd:element name="operationResponse"
               type="OperationResponse"/>
  <xsd:element name="errorNotice"
               type="StatusNotice"/>
 </xsd:choice>
</xsd:complexType>

<xsd:element name="Request" type="Request"/>
<xsd:element name="Response" type="Response"/>
<xsd:element name="LTAPRequest" type="LTAPRequest"/>
<xsd:element name="LTAPResponse" type="LTAPResponse"/>

<xsd:complexType name="LTAPResponse">
 <xsd:choice>
  <xsd:element name="response" type="Response"/>
  <xsd:element name="signedResponse" type="ds:Signature"/>
  <xsd:element name="encryptedResponse" type="enc:EncryptedData"/>
 </xsd:choice>
</xsd:complexType>

<xsd:complexType name="LTAPRequest">
 <xsd:choice>
  <xsd:element name="request" type="Request"/>
   <xsd:element name="signedRequest" type="xd:Signature"/>
```

```
      <xsd:element name="encryptedRequest" type="enc:EncryptedData"/>
    </xsd:choice>
  </xsd:complexType>
```

Authors' Addresses

TOC

	Aleksej Jerman Blazic
	SETCCE
	Jamova 39
	Ljubljana SI-1000
	SLOVENIA
Fax:	+386 1 4773861
Email:	aljosa@setcce.org
	Peter Sylvester
	Groupe ON-X - EdelWeb Project
	15 quai de Dion-Bouton
	Puteaux F-92816
	FRANCE
Fax:	+33 1 40 99 14 14
Email:	Peter.Sylvester@edelweb.fr
	Carl Wallace
	Cygnacom Solutions
	Suite 5200
	7925 Jones Branch Drive
	McLean, VA 22102
Email:	cwallace@cygnacom.com