Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79. This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.
Abstract

In many scenarios, users must be able to demonstrate the (time of) existence, integrity and validity of data including signed data for long or undetermined periods of time. This document specifies XML syntax and processing rules for creating evidence for long-term non-repudiation of existence and integrity of data. ERS-XML provides alternative syntax and processing rules to the ASN.1 ERS [RFC4998] syntax by using XML language.
Conventions used in this document

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].
# Table of Contents

1. Introduction .......................................................... 6
   1.1. Motivation .................................................... 6
   1.2. General Overview and Requirements .......................... 8
   1.3. Terminology .................................................. 9
   1.4. Conventions Used in This Document .......................... 11
2. Evidence Record .................................................... 11
   2.1. Structure ................................................... 12
   2.2. Generation .................................................. 17
   2.3. Verification ................................................ 18
3. Archive Time-Stamp ................................................ 18
   3.1. Structure ................................................... 19
      3.1.1. Hash Tree ............................................ 19
      3.1.2. Time-Stamp .......................................... 20
      3.1.3. Cryptographic Information List ...................... 22
   3.2. Generation .................................................. 23
      3.2.1. Generation of Hash Tree ............................. 23
      3.2.2. Reduction of hash tree ............................. 27
   3.3. Verification ................................................ 29
4. Archive Time-Stamp Sequence and Archive Time-Stamp Chain ...... 30
   4.1. Structure ................................................... 31
      4.1.1. Digest Method ....................................... 32
      4.1.2. Canonicalization Method ........................... 32
   4.2. Generation .................................................. 33
      4.2.1. Time-Stamp Renewal .................................. 34
      4.2.2. Hash Tree Renewal .................................. 35
   4.3. Verification ................................................ 37
5. Encryption .......................................................... 39
6. Storage of policies ................................................ 40
7. XSD Schema for the Evidence Record ................................ 40
8. Security Considerations .......................................... 45
   8.1. Secure Algorithms ......................................... 45
   8.2. Redundancy ................................................ 46
   8.3. Secure Time-Stamps ......................................... 46
1. Introduction

The purpose of the document is to define XML Schema and processing rules for Evidence Record Syntax in XML (Extensible Markup Language) format. The document is related to initial ASN.1 (Abstract Syntax Notation One) syntax for Evidence Record Syntax as defined in [RFC4998].

1.1. Motivation

The evolution of electronic commerce and electronic data exchange in general requires introduction of non-repudiable proof of data existence as well as data integrity and authenticity. Such data and non-repudiable proof of existence must endure for long periods of time, even when the initial information to prove its existence and integrity weakens or ceases to exist. Mechanisms such as digital signatures defined in [RFC5652], for example, do not provide absolute reliability on a long term basis. Algorithms and cryptographic material used to create a signature can become weak in the course of time and information needed to validate digital signatures may become compromised or simply cease to exist due to for example the disbanding of a certificate service provider. Providing a stable environment for electronic data on a long term basis requires the introduction of additional means to continually provide an appropriate level of trust in evidence on data existence, integrity and authenticity.

All integrity and authenticity protecting techniques used today suffer from the problem of degrading reliability over time, including techniques for Time-Stamping, which are generally recognized as data existence and integrity proof mechanisms. Over long periods of time cryptographic algorithms used may become weak or encryption keys compromised. Some of the problems might not even be of technical nature like a Time-Stamping authority going out of business and
ceasing its service. To create a stable environment where proof of existence and integrity can endure well into the future a new technical approach must be used.

Long term non-repudiation of data existence and demonstration of data integrity techniques have been already introduced, for example, by long term signature syntaxes like those defined in [RFC5126]. Long term signature syntaxes and processing rules address only the long term endurance of the digital signatures themselves, while Evidence Record Syntax broadens this approach for data of any type or format including digital signatures.

The XMLERS (Extensible Markup Language Evidence Record Syntax) syntax is based on Evidence Record Syntax as defined in [RFC4998] and is addressing the same problem of long term non-repudiable proof of data existence and demonstration of data integrity on a long term basis. XMLERS does not supplement the [RFC4998] specification. Following extensible markup language standards and [RFC3470] guidelines it introduces the same approach but in a different format and with adapted processing rules.

The use of eXtensible Markup Language (XML) format is already recognized by a wide range of applications and services and is being selected as the de-facto standard for many applications based on data exchange. The introduction of Evidence Record Syntax in XML format broadens the horizon of XML use and presents a harmonized syntax with a growing community of XML based standards including those related to security services such as [XMLDSig] or [XAdES].

Due to the differences in XML processing rules and other characteristics of XML language, XMLERS does not present a direct transformation of ERS in ASN.1 syntax. The XMLERS syntax is based on different processing rules as defined in [RFC4998] and it does not support, for example, the import of ASN.1 values in XML tags. Creating Evidence Records in XML syntax must follow the steps as
defined in this draft. XMLERS is a standalone draft and is based on
[RFC4998] conceptually only.

Evidence Record Syntax in XML format is based on long term archive
service requirements as defined in [RFC4810]. XMLERS syntax delivers
the same (level of) non-repudiable proof of data existence as ASN.1
ERS[RFC4998]. The XML syntax supports archive data grouping (and de-
grouping) together with simple or complex Time-Stamp renewal
processes. Evidence Records can be embedded in the data itself or
stored separately as a standalone XML file.

1.2. General Overview and Requirements

XMLERS specifies the XML syntax and processing rules for creating
evidence for the long-term non-repudiation of existence and integrity
of data in a unit called the "Evidence Record". The XMLERS syntax is
defined to meet the requirements for data structures as set out in
[RFC4810]. This document also refers to the ASN.1 ERS specification
as defined in [RFC4998].

An Evidence Record may be generated and maintained for a single data
object or a group of data objects that form an archive object. A data
object (binary chunk or a file) may represent any kind of document or
part of it. Dependencies among data objects, their validation or any
other relationship than "a data object is a part of particular
archived object" are outside the scope of this draft.

Evidence Record is closely related to Time-Stamping techniques.
However, Time-Stamps as defined in [RFC3161], can cover only a single
unit of data and do not provide processing rules for maintaining a
long term stability of Time-Stamps applied over a data object.
Evidence for an archive object is created by acquiring a Time-Stamp
from a trustworthy authority for a specific value that is
unambiguously related to a single or more data objects. Relationship
between several data objects and a single time-stamped value is
addressed using a hash tree, a technique first described by Merkle
[MER1980] and later in [RFC4998], with data structures and procedures as specified in this document. The Evidence Record Syntax enables processing of several archive objects within a single processing pass using a hash tree technique and acquiring only one Time-Stamp to protect all archive objects.

Besides a Time-Stamp other artifacts are also preserved in Evidence Record: data necessary to verify the relationship between a time-stamped value and a specific data object, packed into a structure called a "hash tree"; and long term proofs for the formal verification of included Time-Stamp(s).

Due to the fact that digest algorithms or cryptographic methods used may become weak or that certificates used within a Time-Stamp (and signed data) may be revoked or expire, the collected evidence data must be monitored and renewed before such events occur. This document introduces XML based syntax and processing rules for the creation and continuous renewal of evidence data.

1.3. Terminology

Archive data object: Data unit that is archived and has to be preserved for a long time by the Long-term Archive Service.

Archive data object group: A set of archive data objects, which for some reason (logically) belong together, e.g. a group of document files or a document file and a signature file could represent an archive data object group.

Archive object: an archive data object or an archive data object group.

Archive Time-Stamp (ATS): An Archive Time-Stamp contains a Time-Stamp Token, useful data for validation and optionally a set of ordered lists of hash values (a hash tree). An Archive Time-Stamp relates to a data object, if the hash value of this data object is part of the
first hash value list of the Archive Time-Stamp or its hash value matches the time-stamped value. An Archive Time-Stamp relates to a data object group, if it relates to every data object of the group and no other data object (i.e. the hash values of all but no other data objects of the group are part of the first hash value list of the Archive Time-Stamp) (see section 3.).

Archive Time-Stamp Chain (ATSC): holds a sequence of Archive Time-Stamps generated during the preservation period.

Archive Time-Stamp Sequence (ATSSeq): is a sequence of Archive Time-Stamp Chains.

Canonicalization: Processing rules for transforming an XML document into its canonical form. Two XML documents may have different physical representations, but they may have the same canonical form. For example a sort order of attributes does not change the meaning of the document as defined in [XMLC14N].

Cryptographic Information: Data or part of data related to the validation process of signed data, e.g. digital certificates, digital certificate chains, certificate revocation lists, etc.

Digest Method: Digest method is a digest algorithm, which is a strong one-way function, for which it is computationally infeasible to find an input that corresponds to a given output or to find two different input values that correspond to the same output. A digest algorithm transforms input data into a short value of fixed length. The output is called digest value, hash value or data fingerprint.

Evidence: Information that may be used to resolve a dispute about various aspects of authenticity, validity and existence of archived data objects.
Evidence Record: Collection of evidence compiled for a given archive object over time. An Evidence Record includes ordered collection of ATSS, which are grouped into ATSCs and ATSSeqs.

Long-term Archive Service (LTA): A service responsible for generation, collection and maintenance (renewal) of evidence data. A LTA service may also preserve data for long periods of time, e.g. storage of archive data and associated evidences.

Hash Tree: Collection of hash values of protected objects (input data objects and generated evidence within archival period) that are unambiguously related to the time-stamped value within an Archive Time-Stamp.

Time-Stamp Token (TS): A cryptographically secure confirmation generated by a Time-Stamping Authority (TSA) e.g. [RFC3161] which specifies a structure for Time-Stamps and a protocol for communicating with a Time-Stamp Authority. Besides this, other data structures and protocols may also be appropriate, such as defined in [ISO-18014-1.2002], [ISO-18014-2.2002], [ISO-18014-3.2004], and [ANSI.X9-95.2005].

1.4. Conventions Used in This Document

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].

2. Evidence Record

An Evidence Record is a unit of data that is to be used to prove the existence of an archive object (a single archive data object or a archive data object group) at a certain time. Through the lifetime of an archive object, an Evidence Record also demonstrates the data objects’ integrity and non-repudiability. To achieve this, cryptographic means are used, i.e. the LTA obtains Time-Stamp Tokens.
from the Time-Stamping Authority (TSA). It is possible to store the Evidence Record separately from the archive object or to integrate it into the data itself.

As cryptographic means are used to support Evidence Records, such records may lose their value over time. Time-Stamps obtained from Time-Stamping Authorities may become invalid for a number of reasons, usually due to time constraints of Time-Stamp validity or when cryptographic algorithms lose their security properties. Before the used Time-Stamp Tokens become unreliable, the Evidence Record has to be renewed. This may result in a series of Time-Stamp Tokens, which are linked between themselves according to the cryptographic methods and algorithms used.

Evidence Records can be supported with additional information, which can be used to ease the processes of Evidence Record validation and renewal. Information such as digital certificates and certificate revocation lists as defined in [RFC5280] or other cryptographic material can be collected, enclosed and processed together with archive object data (i.e. time-stamped).

2.1. Structure

The Evidence Record contains one or several Archive Time-Stamps (ATS). An ATS contains a Time-Stamp Token and optionally other useful data for Time-Stamp validation, e.g. certificates, CRLs (certificate revocation list) or OCSP (Online Certificate Status Protocol) responses and also specific attributes such as service policies. Initially, an ATS is acquired and later, before it expires or becomes invalid a new ATS is acquired, which prolongs the validity of the archived object (its data objects together with all previously generated Archive Time-Stamps). This process must continue during the desired archiving period of the archive data object(s). A series of successive Archive Time-Stamps is collected in Archive Time-Stamp Chains and a series of chains in Archive Time-Stamp Sequence.
In XML syntax the Evidence Record is represented by the 
<EvidenceRecord> root element, which has the following structure 
(where "?" denotes zero or one occurrences, "+" denotes one or more 
ocurrences and "*" denotes zero or more occurrences):

<EvidenceRecord>
  <EncryptionInformation>
    <EncryptionInformationType>
      <EncryptionInformationValue>
    </EncryptionInformationValue>
  </EncryptionInformation> ?
  <SupportingInformationList>
    <SupportingInformation Type /> +
  </SupportingInformation> ?
  <ArchiveTimeStampSequence>
    <ArchiveTimeStampChain Order>
      <DigestMethod Algorithm />
      <CanonicalizationMethod Algorithm />
    </ArchiveTimeStamp>
    <TimeStamp>
      <TimeStampToken Type />
      <CryptographicInformationList>
        <CryptographicInformation Order Type /> +
      </CryptographicInformationList>
    </TimeStamp>
    <Attributes>
      <Attribute Order Type /> +
    </Attributes> ?
  </ArchiveTimeStampChain> +
</ArchiveTimeStampSequence>
</EvidenceRecord>

The syntax of an evidence record is defined as an XML schema [XMLSchema], see Section 6. The schema uses the following XML
namespace urn:ietf:params:xml:ns:ers as default namespace and starts with following definition:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns ="urn:ietf:params:xml:ns:ers"
         targetNamespace="urn:ietf:params:xml:ns:ers"
         elementFormDefault="qualified"
         attributeFormDefault="unqualified">
```

The XML tags have the following meanings:

Version attribute indicates the syntax version, for compatibility with future revisions of this specification and to distinguish it from earlier non-conformant or proprietary versions of the XMLERS. Current version of the XMLERS syntax is 1.0.

<EncryptionInformation> tag is optional and holds information on cryptographic algorithms and cryptographic material used to encrypt archive data (in case archive data is encrypted e.g. for privacy purposes). This optional information is needed to unambiguously re-encrypt data objects when processing Evidence Records. When omitted, data objects are not encrypted or non-repudiation proof is not needed for the unencrypted data. Details on how to process encrypted archive data and generate Evidence Record(s) are described in Section 5.

<SupportingInformationList> tag is optional and can hold information to support processing of Evidence Records. An example of this supporting information may be a processing policy, like a cryptographic policy (e.g. [RFC5698]) or archiving policies, which can provide input about preservation and evidence validation. Each data object is put into a separate child element <SupportingInformation>, with an optional Type attribute to indicate its type for processing directions. Note that if supporting information and policies are relevant for and already available at or before the time of individual renewal steps (e.g.
to indicate the DSSC crypto policy ([RFC5698]) that was used at the time of the individual renewal) they SHOULD be stored in the <Attributes> element of the individual Archive Time-Stamp (see below) as this is integrity protected by the Archive Time-Stamps. Supporting information that is relevant for the whole Evidence Record (like the LTA’s current Cryptographic Algorithms Security Suitability policy (DSSC, [RFC5698]) or that was not available at the time of renewal (and therefore could not later be stored in the protected <Attributes> tag, can be stored in this <SupportingInformation> tag.

<ArchiveTimeStampSequence> is a sequence of <ArchiveTimeStampChain>.

<ArchiveTimeStampChain> holds a sequence of Archive Time-Stamps generated during the preservation period. Details on Archive Time-Stamp Chains and Archive Time-Stamp Sequences are described in section 4. The sequences of Archive Time-Stamp Chains and Archive Time-Stamps are ordered and the order must be indicated with "Order" attribute of the <ArchiveTimeStampChain> and <ArchiveTimeStamp> element.

<DigestMethod> is a required element and identifies the digest algorithm used within one Archive Time-Stamp chain to calculate digest values from archive data object(s), previous Archive Time-Stamp sequence, Time-Stamps and within a Time-Stamp Token.

<CanonicalizationMethod> is a required element that specifies which canonicalization algorithm is applied to the archive data for XML data objects, <ArchiveTimeStampSequence> or <TimeStamp> elements prior to performing digest value calculations.

<HashTree> tag holds a structure as described in section 3.1.1.
<TimeStamp> tag holds a <TimeStampToken> element with a Time-Stamp Token provided by the Time-Stamping Authority and optional element <CryptographicInformationList>.

<CryptographicInformationList> tag allows the storage of data needed in the process of Time-Stamp Token validation in case when such data is not provided by the Time-Stamp Token itself. This could include possible trust anchors, certificates, revocation information or the current definition of the suitability of cryptographic algorithms, past and present. Each data object is put into a separate child element <CryptographicInformation>, with a mandatory Order attribute to indicate the order within its parent element. These items may be added based on the policy used. This data is protected by successive Time-Stamps in the sequence of the Archive Time-Stamps.

<Attributes> tag contains additional information that may be provided by an LTA used to support processing of Evidence Records. An example of this supporting information may be a processing policy, like a renewal, a cryptographic (e.g. [RFC5698]) or an archiving policy. Such policies can provide inputs, which are relevant for data object(s) preservation and evidence validation at a later stage. Each data object is put into a separate child element <Attribute>, with a mandatory Order attribute to indicate the order within the parent element and an optional Type attribute to indicate processing directions.

The Order attribute is mandatory in all cases when one or more XML elements with the same name occur on the same level in the XMLERS structure. Although most of the XML parsers will preserve the order of the sibling elements having the same name, within XML structure there is no definition how to unambiguously define such order. Preserving the correct order in such cases is of significant importance for digest value calculations over XML structures.
2.2. Generation

The generation of an <EvidenceRecord> element can be described as follows:

1. Select an archive object (a data object or a data object group) to archive.

2. Create the initial <ArchiveTimeStamp>. This is the first ATS within the initial <ArchiveTimeStampChain> element of the <ArchiveTimeStampSequence> element.

3. Refresh the <ArchiveTimeStamp> when necessary by Time-Stamp Renewal or Hash Tree Renewal (see Section 4.).

The Time-Stamping service may be, for a large number of archived objects, expensive and time-demanding, so the LTA may benefit from acquiring one Time-Stamp Token for many archived objects, which are not otherwise related to each other. It is possible to collect many archive objects, build a hash tree to generate a single value to be time-stamped, and respectively reduce that hash tree to small subsets that for each archive object provide necessary binding with the time-stamped hash value (see Section 3.2.1).

For performance reasons or in case of local Time-Stamp generation, building a hash tree (<HashTree> element) can be omitted. It is also possible to convert existing Time-Stamps into an ATS for renewal.

The case when only essential parts of documents or objects shall be protected is out of scope for this standard, and an application that is not defined in this draft must ensure that the correct unambiguous extraction of binary data is made for the generation of Evidence Record.

An application may also provide evidence such as certificates, revocation lists etc., needed to verify and validate signed data.
objects or a data object group. This evidence may be added to the archived object data group and will be protected within initial (and successive) Time-Stamp(s).

Note that the <CryptographicInformationList> element of Evidence Record is not to be used to store and protect cryptographic material related to signed archive data. The use of this element is limited to cryptographic material related to TS(s).

2.3. Verification

The overall verification of an Evidence Record can be described as follows:

1. Select an archive object (a data object or a data object group)
2. Re-encrypt data object or data object group, if encryption field is used (for details, see Section 5.).
3. Verify Archive Timestamp Sequence (details in Section 3.3. and Section 4.3.).

3. Archive Time-Stamp

An Archive Time-Stamp is a timestamp with additional artifacts that allow the verification of the existence of several data objects at a certain time.

The process of construction of an ATS must support evidence on a long term basis and prove that the archive object existed and was identical, at the time of the Time-Stamp, to the currently present archive object (at the time of verification). To achieve this, an ATS must be renewed before it becomes invalid (which may happen for several reasons such as e.g. weakening used cryptographic algorithms, invalidation of digital certificate or decomposing TSA).
3.1. Structure

An Archive Time-Stamp contains a Time-Stamp Token, with useful data for its validation (cryptographic information), such as the certificate chain or certificate revocation lists, an optional ordered set of ordered lists of hash values (a hash tree) that were protected with the Time-Stamp Token and optional information describing the renewal steps (<Attributes> element). A hash tree may be used to store data needed to bind the time-stamped value with protected objects by the Archive Time-Stamp. If a hash tree is not present, the ATS simply refers to a single object; either input data object or a previous TS.

3.1.1. Hash Tree

Hash tree structure is an optional container for significant values, needed to unambiguously relate a time-stamped value to protected data objects, and is represented by the <HashTree> element. The root hash value that is generated from the values of the hash tree MUST be the same as the time-stamped value.

<HashTree>
  <Sequence Order>
    <DigestValue>base64 encoded hash value</DigestValue> +
  </Sequence> +
</HashTree>

The algorithm by which a root hash value is generated from the <HashTree> element is as follows: the content of each <DigestValue> element within the first <Sequence> element is base64 decoded to obtain a binary value (representing the hash value). All collected hash values from the sequence are ordered in binary ascending order, concatenated and a new hash value is generated from that string. With one exception from this rule: when the first <Sequence> element has
only one <DigestValue> element, then its binary value is added to the next list obtained from the next <Sequence> element. The newly calculated hash value is added to the next list of hashes obtained from the next <Sequence> element and the previous step is repeated until there is only one hash value left, i.e. when there are no <Sequence> elements left. The last calculated hash value is the root hash value. When an archive object is a group and composed of more than one data object, the first hash list MUST contain the hash values of all its data objects.

When a single Time-Stamp is obtained for a set of archive objects, the LTA MUST construct a hash tree to generate a single hash value to bind all archive objects from that group and then a reduced hash tree MUST be calculated from the hash tree for each archive object respectively (see Section 3.2.1).

For example: A SHA-1 digest value is a 160-bit string. The text value of the <DigestValue> element shall be the base64 encoding of this bit string viewed as a 20-octet octet stream. And to continue the example, the text value of a <DigestValue> element for the message digest A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D would be:

<DigestValue>qZk+NkcGgWq6PiVxeFDCbJzQ2J0=</DigestValue>

3.1.2. Time-Stamp

Time-Stamp Token is an attestation generated by a TSA that a data item existed at a certain time. The Time-Stamp Token is a signed data object that contains the hash value, the identity of the TSA, and the exact time (obtained from trusted time source) of Time-Stamping. This proves that the given data existed before the time of Time-Stamping. For example, [RFC3161] specifies a structure for signed Time-Stamp Tokens in ASN.1 format. Since at the time being there is no standard for an XML Time-Stamp, the following structure example is provided (referring to the Entrust XML Schema for Time-Stamp http://www.entrust.com/schemas/timestamp19protocol-20020207), which
is a digital signature compliant to [XMLDSig] specification containing Time-Stamp specific data, such as time-stamped value and time within <Object> element of a signature.

    <element name="TimeStampInfo">
      <complexType>
        <sequence>
          <element ref="ts:Policy" />  
          <element ref="ts:Digest" /> 
          <element ref="ts:SerialNumber" minOccurs="0" /> 
          <element ref="ts:CreationTime" /> 
          <element ref="ts:Accuracy" minOccurs="0" /> 
          <element ref="ts:Ordering" minOccurs="0" /> 
          <element ref="ts:Nonce" minOccurs="0" /> 
          <element ref="ts:Extensions" minOccurs="0" /> 
          </sequence>
      </complexType>
    </element>

A <TimeStamp> element of ATS holds a complete structure of Time-Stamp Token as provided by a TSA. Time-Stamp Token may be in XML or ASN.1 format. The Attribute type MUST be used to indicate the format for processing purposes, with values XMLENTRUST or RFC3161 respectively. For an RFC3161 type Time-Stamp Token, the <TimeStamp> element MUST contain base64 encoding of a DER-encoded ASN1 data. These type values are registered by IANA (see Section 8). For support of future types of timestamps (in particular for future XML time-stamp standards), these need to be registered there as well.

For example:

    <TimeStamp Type="RFC3161">MIAGCSqGSIb3DQE...</TimeStamp>

or
<TimeStamp Type="XMLENTRUST"><dsig:Signature>...</dsig:Signature></TimeStamp>.

3.1.3. Cryptographic Information List

Digital certificates, CRLs, SCVP or OCSP-Responses needed to verify the Time-Stamp Token SHOULD be stored in the Time-Stamp Token itself. When this is not possible, such data MAY be stored in the <CryptographicInformationList> element, each data object into a separate <CryptographicInformation> element, using a mandatory Order attribute.

The attribute Type is mandatory and is used to store processing information about the type of stored cryptographic information. The Type attribute MUST use a value registered with IANA, as identifiers: CRL, OCSP, SCVP or CERT, and for each type the content MUST be encoded respectively:

- for type CRL, a base64 encoding of a DER-encoded X.509 CRL [RFC5280];
- for type OCSP, a base64 encoding of a DER-encoded OCSPResponse [RFC2560];
- for type SCVP, a base64 encoding of a DER-encoded CVResponse [RFC5055];
- for type CERT, a base64 encoding of a DER-encoded X.509 certificate [RFC5280].

The supported type identifiers are registered by IANA (see Section 8). Future supported types can be registered there (for example to support future validation standards).
3.2. Generation

An initial ATS relates to a data object or a data object group that represents an archive object. The generation of the initial ATS element can be done in a single process pass for one or for many archived objects, described as follows:

1. Collect one or more archive objects to be time-stamped.

2. Select a canonicalization method C to be used for obtaining binary representation of archive data and for Archive Time-Stamp at a later stage in the renewing process (see section 4). Note that the selected canonicalization method MUST be used also for archive data when data is represented in XML format.

3. Select a valid digest algorithm H. The selected secure hash algorithm MUST be the same as the hash algorithm used in the Time-Stamp Token and for the hash tree computations.

4. Generate a hash tree for selected archive object (see 3.2.1).

   The hash tree may be omitted in the initial ATS, when an archive object has a single data object; then the time-stamped value must match the digest value of that single data object.

5. Acquire Time-Stamp token from TSA for root hash value of a hash tree (see 3.1.1). If the Time-Stamp token is valid, the initial Archive Time-Stamp may be generated.

3.2.1. Generation of Hash Tree

   The <DigestValue> elements within the <Sequence> element MUST be ordered in binary ascending order to ensure the correct calculation of digest values at the time of renewal and later for verification purposes. Note, that the text value of <DigestValue> element is
base64 encoded, so it MUST be base64 decoded in order to obtain a binary representation of the hash value.

A hash tree MUST be generated when the time-stamped value is not equal to the hash value of the input data object. This is the case when either of the following is true:

1. When an archive object has more than one data object, its digest value is the digest value of binary ascending ordered and concatenated digest values of all its containing data objects. Note that in this case the first list of the hash tree MUST contain hash values of all data objects and only those values.

2. When for more than one archive object a single Time-Stamp Token is generated, then the hash tree is a reduced hash tree extracted from hash tree for that archive object (see Section 3.2.2).

The hash tree for a group of archive objects is built from the leaves to the root. First the leaves of the tree are collected, each leaf representing the digest value of an archive object. Use the following procedure to calculate the hash tree:

1. Collect archive objects and for each archive object its corresponding data objects.

2. Choose a secure hash algorithm H and calculate the digest values for the data objects and put them into the input list for the hash tree as follows: a digest value of an archive object is the digest value of its data object, if there is only one data object; for more than one data object a digest value is the digest value of binary sorted, concatenated digest values of all its containing data objects.
Note that for archive objects having more than one data object, lists of their sub-digest values are stored and later, when creating a reduced hash tree for that archive object, they will became members of the first hash list.

3. Group together items in the input list by the order of N (e.g. for a binary tree group in pairs) and for each group: binary ascending sort, concatenate and calculate hash values. The result is a new input list.

4. Repeat step 3, until only one digest value is left; this is the root value of the hash tree, which is time-stamped.

Note that the selected secure hash algorithm MUST be the same as the one defined in the <DigestMethod> element of the ATSChain.

Example: An input list with 18 hash values, where the h’1 is generated for a group of data objects (d4, d5, d6 and d7) and has been grouped by 3. The group could be of any size (2, 3...). It is also possible to extend the tree with "dummy" values; to make every node have the same number of children.
Figure 1 Generation of the Merkle Hash Tree.
Note that there are no restrictions on the quantity of hash value lists and of their length. Also note that it is beneficial but not required to build hash trees and reduce hash trees. An Archive Time-Stamp may consist only of one list of hash values and a Time-Stamp or in an extreme case only a Time-Stamp with no hash value lists.

3.2.2. Reduction of hash tree

The generated Merkle hash tree can be reduced to lists of hash values, necessary as a proof of existence for a single archive object as follows:

1. For a selected archive object (AO) select its hash value h within the leaves of the hash tree.

2. Put h as base64 encoded text value of a new <DigestValue> element within a first <Sequence> element. If the selected archive object AO is a data object group (i.e. has more than one data object), the first <Sequence> element MUST in this case be formed from the hash values of all AO’s data objects, each within a separate <DigestValue> element.

3. Select all hash values, which have the same father node as hash value h. Place these hash values each as a base64 encoded text value of a new <DigestValue> element within a new <Sequence> element, increasing its Order attribute value by 1.

4. Repeat step 3 for the parent node until the root hash value is reached, with each step create a new <Sequence> element and increase its Order attribute by one. Note that node values are not saved as they are computable.

The order of <DigestValue> elements within each <Sequence> element MUST be binary ascending (by base64 decoded values).
Reduced hash tree for data object d4 (from the previous example, presented in Figure 1):

<HashTree>
  <Sequence Order='1'>
    <DigestValue>base64 encoded h4</DigestValue>
    <DigestValue>base64 encoded h5</DigestValue>
    <DigestValue>base64 encoded h6</DigestValue>
    <DigestValue>base64 encoded h7</DigestValue>
  </Sequence>
  <Sequence Order='2'>
    <DigestValue>base64 encoded h8</DigestValue>
    <DigestValue>base64 encoded h9</DigestValue>
  </Sequence>
  <Sequence Order='3'>
    <DigestValue>base64 encoded h''1</DigestValue>
    <DigestValue>base64 encoded h''3</DigestValue>
  </Sequence>
  <Sequence Order='4'>
    <DigestValue>base64 encoded h''2</DigestValue>
  </Sequence>
</HashTree>

Reduced Hash tree for data object d2 (from the previous example, presented in Figure 1):

<HashTree>
  <Sequence Order='1'>
    <DigestValue>base64 encoded h2</DigestValue>
  </Sequence>
  <Sequence Order='2'>
    <DigestValue>base64 encoded h1</DigestValue>
    <DigestValue>base64 encoded h3</DigestValue>
  </Sequence>
  <Sequence Order='3'>
    <DigestValue>base64 encoded h''2</DigestValue>
  </Sequence>
</HashTree>
3.3. Verification

The initial Archive Timestamp shall prove that an archive object existed at a certain time, indicated by its Time-Stamp token. The verification procedure is as follows:

1. Identify hash algorithm H (from <DigestMethod> element) and calculate the hash value for each data object of the archive object.

2. If the hash tree is present, search for hash values in the first <Sequence> element. If hash values are not present, terminate verification process with negative result. If the verifying party also seeks additional proof that the Archive Time-Stamp relates to a data object group (e.g. a document and all its digital signatures), it can also be verified that only the hash values of the data objects that are members of the given data object group are in the first hash value list.

3. If the hash tree is present, calculate its root hash value. Compare the root hash value with the time-stamped value. If not equal, terminate verification process with negative result.

4. If the hash tree is omitted, compare the hash value of the single data object with the time-stamped value. If not equal, terminate verification process with negative result. If an archive object is having more data objects and the hash tree is omitted, also exit with negative result.
5. Check the validity of the Time-Stamp token. If the needed information to verify formal validity of the Time-Stamp token is not available or found within the <TimeStampToken> element or within <CryptographicInformationList> element or in <SupportingInformationList> (see section 7), exit with a negative result.

Information for formal verification of the Time-Stamp token includes digital certificates, Certificate Revocation Lists, Online Certificate Status Protocol responses, etc. This information needs to be collected prior to the Time-Stamp renewal process and protected with the succeeding Time-Stamp, i.e. included in the <TimeStampToken> or <CryptographicInformation> element (see section 8 for additional information and section 4.2.1 for details on Time-Stamp renewal process). For the current (latest) Time-Stamp, information for formal verification of the (latest) Time-Stamp should be provided by the Time-Stamping Authority. This information can also be provided with the Evidence Record within <SupportingInformation> element, which is not protected by any Time-Stamp.

4. Archive Time-Stamp Sequence and Archive Time-Stamp Chain

An Archive Time-Stamp proves the existence of single data objects or a data object group at a certain time. However, the initial Evidence Record created can become invalid due to loosing the validity of the Time-Stamp Token for a number of reasons: hash algorithms or public key algorithms used in its hash tree or the Time-Stamp may become weak or the validity period of the timestamp authority certificate expires or is revoked.

To preserve the validity of an Evidence Record before such events occur, the Evidence Record has to be renewed. This can be done by creating a new ATS. Depending on the reason for renewing the Evidence Record (the Time-Stamp becomes invalid or the hash algorithm of the hash tree becomes weak) two types of renewal processes are possible:
o Time-Stamp renewal: For this process a new Archive Time-Stamp is generated, which is applied over the last Time-Stamp created. The process results in a series of Archive Time-Stamps which are contained within a single Archive Time-Stamp Chain (ATSC).

o Hash Tree renewal: For this process a new Archive Time-Stamp is generated, which is applied to all existing Time-Stamps and data objects. The newly generated Archive Time-Stamp is placed in a new Archive Time-Stamp Chain. The process results in a series of Archive Time-Stamp Chains which are contained within a single Archive Time-Stamp Sequence (ATSS).

After the renewal process, only the most recent (i.e. the last generated) Archive Time-Stamp has to be monitored for expiration or validity loss.

4.1. Structure

Archive Time-Stamp Chain and Archive Time-Stamp Sequence are containers for sequences of archive Time-Stamps which are generated through renewal processes. The renewal process results in a series of Evidence Record elements: 

- `<ArchiveTimeStampSequence>` element contains an ordered sequence of `<ArchiveTimeStampChain>` elements and
- `<ArchiveTimeStampChain>` element contains an ordered sequence of `<ArchiveTimeStamp>` elements. Both elements MUST be sorted by time of the Time-Stamp in ascending order. Order is indicated by the Order attribute.

When an Archive Time-Stamp must be renewed, a new `<ArchiveTimeStamp>` element is generated and depending on the generation process, it is either placed:

- as the last `<ArchiveTimeStamp>` child element in a sequence of the last `<ArchiveTimeStampChain>` element in case of Time-Stamp renewal or
o as the first <ArchiveTimeStamp> child element in a sequence of the newly created <ArchiveTimeStampChain> element in case of hash tree renewal.

The ATS with the largest Order attribute value within the ATSC with the largest Order attribute value is the latest ATS and MUST be valid at the present time.

4.1.1. Digest Method

Digest method is a required element that identifies the digest algorithm used to calculate hash values of archive data (and node values of hash tree). The digest method is specified in the <ArchiveTimeStampChain> element by the required <DigestMethod> element and indicates the digest algorithm that MUST be used for all hash value calculations related to the Archive Time-Stamps within the Archive Time-Stamp chain.

The Algorithm attribute contains URIs for identifiers which MUST be used as defined in [RFC3275] and [RFC4051]. For example when the SHA-1 algorithm is used, the algorithm identifier is:

<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>

Within a single ATSC the digest algorithms used for the hash trees of its Archive Time-Stamps and the Time-Stamp Token(s) MUST be the same. When algorithms used by a TSA are changed (e.g. upgraded) a new ATSC MUST be started using an equal or stronger digest algorithm.

4.1.2. Canonicalization Method

Prior to hash value calculations of an XML element, a proper binary representation must be extracted from its (abstract) XML data presentation. The binary representation is determined by UTF-8 encoding and canonicalization of the XML element. The XML element includes the entire text of the start and end tags as well as all
descendant markup and character data (i.e. the text and sub-elements) between those tags.

<CanonicalizationMethod> is a required element that identifies the canonicalization algorithm used to obtain binary representation of an XML element(s). Algorithm identifiers (URIs) must be used as defined in [RFC3275] and [RFC4051]. For example when Canonical XML 1.0 (omits comments) is used, algorithm identifier is

<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>

Canonicalization MUST be applied over XML structured archive data and MUST be applied over elements of Evidence Record (namely ATS and ATSC in the renewing process).

The canonicalization method is specified in the <Algorithm> attribute of the <CanonicalizationMethod> element within the <ArchiveTimeStamChain> element and indicates the canonicalization method that MUST be used for all binary representations of the Archive Time-Stamps within that Archive Time-Stamp chain. In case of succeeding ATSC the canonicalization method indicated within the ATSC must also be used for the calculation of the digest value of the preceding ATSC. Note that the canonicalization method is unlikely to change over time as it does not impose the same constraints as the digest method. In theory, the same canonicalization method can be used for a whole Archive Time-Stamp Sequence. Although alternative canonicalization methods may be used, it is recommended to use c14n-20010315 [XMLC14N].

4.2. Generation

Before the cryptographic algorithms used within the most recent Archive Time-Stamp become weak or the Time-Stamp certificates are invalidated, the LTA has to renew the Archive Time-Stamps by
generating a new Archive Time-Stamp using one of two procedures: time-stamp renewal or hash tree renewal.

4.2.1. Time-Stamp Renewal

In case of Time-Stamp renewal, i.e. if the digest algorithm (H) to be used in the renewal process is the same as digest algorithm (H’) used in the last Archive Time-Stamp, the complete content of the last <TimeStamp> element MUST be time-stamped and new <ArchiveTimeStamp> element created as follows:

1. If the current <ArchiveTimeStamp> element does not contain needed proof for long-term formal validation of its Time-Stamp Token within the <TimeStamp> element, collect needed data such as root certificates, certificate revocation lists, etc., and include them in <CryptographicInformationList> element of the last Archive Time-Stamp (each data object into a separate <CryptographicInformation> element).

2. Select canonicalization method from <CanonicalizationMethod> element and select digest algorithm from <DigestMethod> element. Calculate hash value from binary representation of the <TimeStamp> element of the last <ArchiveTimeStamp> element including added cryptographic information. Acquire the Time-Stamp for the calculated hash value. If the Time-Stamp is valid, the new Archive Time-Stamp may be generated.

3. Increase the value order of the new ATS by one and place the new ATS into the last <ArchiveTimeStampChain> element.

The new ATS and its hash tree MUST use the same digest algorithm as the preceding one, which is specified in the <DigestMethod> element within <ArchiveTimeStampChain> element. Note that the new ATS MAY not contain a hash tree. However, Time-Stamp Renewal process may be optimized to acquire one Time-Stamp for many Archive Time-Stamps.
using a hash tree. Note that each hash of the <TimeStamp> element is treated as the document hash in Section 3.2.1.

4.2.2. Hash Tree Renewal

The process of hash tree renewal occurs when the new digest algorithm is different to the one used in the last Archive Time-Stamp (H ↔ H’). In this case the complete Archive Time-Stamp Sequence and the archive data objects covered by existing Archive Time-stamp must be time-stamped as follows:

1. Select one or more archive objects to be renewed and their current <ArchiveTimeStamp> elements.

2. For each archive object check the current <ArchiveTimeStamp> element. If it does not contain the proof needed for long-term formal validation of its Time-Stamp Token within the Time-Stamp Token, collect the needed data such as root certificates, certificate revocation lists, etc., and include them in the <CryptographicInformationList> element of the last Archive Time-Stamp (each data object into a separate <CryptographicInformation> element).

3. Select a canonicalization method C and select a new secure hash algorithm H.

4. For each archive object select its data objects d(i). Generate hash values h(i) = H(d(i)), for example: h(1), h(2)..., h(n).

5. For each archive object calculate a hash hseq=H(ATSSeq) from binary representation of the <ArchiveTimeStampSequence> element, corresponding to that archive object. Note that Archive Time-Stamp Chains and Archive Time-Stamps MUST be chronologically ordered, each respectively to its Order attribute, and that the canonicalization method C MUST be applied.
6. For each archive object sort in binary ascending order and concatenate all \( h(i) \) and the hseq. Generate a new digest value \( h(j) = H(h(1)\ldots h(n), \text{hseq}) \).

7. Build a new Archive Time-Stamp for each \( h(j) \) (hash tree generation and reduction is defined in sections 3.2.1. and 3.2.2.). Note that each \( h(j) \) is treated as the document hash in section 3.2.1. The first hash value list in the reduced hash tree should only contain \( h(i) \) and hseq.

8. Create the new <ArchiveTimeStampChain> containing the new <ArchiveTimeStamp> element (with order number 1), and place it into the existing <ArchiveTimeStampSequence> as a last child with the order number increased by one.

Example for an archive object with 3 data objects: Select a new hash algorithm and canonicalization method. Collect all 3 data objects and currently generated Archive Time-Stamp sequence.

AO
/ | \
d1 d2 d3

ATSSeq

ATSChain1: ATS0, ATS1

ATSChain2: ATS0, ATS1, ATS2
The hash values MUST be calculated with the new hash algorithm H for all data objects and for the whole ATSSeq. Note, that ATSSeq MUST be chronologically ordered and canonicalized before retrieving its binary representation.

When generating the hash tree for the new ATS, the first sequence become values: H(d1), H(d2),..., H(dn), H(ATSSeq). Note: hash values MUST be sorted in binary ascending order.

```
<HashTree>
  <Sequence Order='1'>
    <DigestValue>H(d1)</DigestValue>
    <DigestValue>H(d2)</DigestValue>
    <DigestValue>H(d3)</DigestValue>
    <DigestValue>H(ATSSeq)</DigestValue>
  </Sequence>
</HashTree>
```

Note that if the group processing is being performed, the hash value of the concatenation of the first sequence is an input hash value into the hash tree.

### 4.3. Verification

An Evidence Record shall prove that an archive object existed and has not been changed from the time of the initial Time-Stamp Token within the first ATS. In order to complete the non-repudiation proof for an archive object, the last ATS has to be valid and ATSCs and their relations to each other have to be proved:
1. Select archive object and re-encrypt its data object or data object group, if <EncryptionInformation> field is used. Select the initial digest algorithm specified within the first Archive Time-Stamp Chain and calculate hash value of the archive object. Verify that the initial Archive Time-Stamp contains (identical) hash value of the AO’s data object (or hash values of AO’s data object group). Note that when the hash tree is omitted, calculated AO’s value MUST match the time-stamped value.

2. Verify each Archive Time-Stamp Chain and each Archive Time-Stamp within. If the hash tree is present within the second and the next Archive Time-Stamps of an Archive Time-Stamp Chain, the first <Sequence> MUST contain the hash value of the <TimeStamp> element before. Each Archive Time-Stamp MUST be valid relative to the time of the succeeding Archive Time-Stamp. All Archive Time-Stamps with the Archive Time-Stamp Chain MUST use the same hash algorithm, which was secure at the time of the first Archive Time-Stamp of the succeeding Archive Time-Stamp Chain.

3. Verify that the first hash value list of the first Archive Time-Stamp of all succeeding Archive Time-Stamp Chains contains hash values of data object and the hash value of Archive Time-Stamp Sequence of the preceding Archive Time-Stamp Chains. Verify that Archive Time-Stamp was created when the last Archive Time-Stamp of the preceding Archive Time-Stamp Chain was valid.

4. To prove the Archive Time-Stamp Sequence relates to a data object group, verify that the first Archive Time-Stamp of the first Archive Time-Stamp Chain does not contain other hash values in its first hash value list than the hash values of those data objects.

For non-repudiation proof for the data object, the last Archive Time-Stamp MUST be valid at the time of verification process.
5. Encryption

In some archive services scenarios it may be required that clients send encrypted data only, preventing information disclosure to third parties, such as archive service providers. In such scenarios it must be clear that Evidence Records generated refer to encrypted data objects. Evidence Records in general protect the bit-stream (or binary representation of XML data) which freezes the bit structure at the time of archiving. Encryption schemes in such scenarios cannot be changed afterwards without losing the integrity proof. Therefore, an ERS record must hold and preserve encryption information in a consistent manner.

Encryption is a two way process, whose result depends on the cryptographic material used, e.g. encryption keys and encryption algorithms. Encryption and decryption keys as well as algorithms must match in order to reconstruct the original message or data that was encrypted. When different cryptographic material is used, the results may not be the same, i.e. decrypted data does not match the original (unencrypted) data. In cases when evidence was generated to prove the existence of encrypted data the corresponding algorithm and decryption keys used for encryption must become a part of the Evidence Record and is used to unambiguously represent original (unencrypted) data that was encrypted.

Cryptographic material may also be used in scenarios when a local copy of encrypted data submitted to the archive service provider for preservation is kept in an unencrypted form by a client. In such scenarios cryptographic material is used to re-encrypt unencrypted data kept by a client for the purpose of performing validation of Evidence Record, which is related to the encrypted form of client’s data.

The attribute Type within <EncryptionInformation> element is optional and is used to store processing information about type of stored encryption information, e.g. encryption algorithm or encryption key.
The use of encryption elements heavily depends on the cryptographic mechanism and has to be defined by other specification.

6. Storage of policies

As explained above policies can be stored in the Evidence Record in the in the <Attribute> or the <SupportingInformation> element. In the case of storing DSSC policies [RFC5698], the types to be used in the <Attribute> or <SupportingInformation> element are defined in [RFC5698, section A.2] for both ASN.1 and XML representation.

7. XSD Schema for the Evidence Record

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:ietf:params:xml:ns:ers"
  targetNamespace="urn:ietf:params:xml:ns:ers"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="EvidenceRecord" type="EvidenceRecordType"/>

  <!-- TYPE DEFINITIONS-->

  <xs:complexType name="EvidenceRecordType">
    <xs:sequence>
      <xs:element name="EncryptionInformation" type="EncryptionInfo" minOccurs="0"/>
      <xs:element name="SupportingInformationList" type="SupportingInformationType" minOccurs="0"/>
      <xs:element name="ArchiveTimeStampSequence" type="ArchiveTimeStampSequenceType"/>
    </xs:sequence>
    <xs:attribute name="Version" type="xs:string" use="required" fixed="1"/>
  </xs:complexType>
</xs:schema>
<xs:complexType name="EncryptionInfo">
  <xs:sequence>
    <xs:element name="EncryptionInfoType" type="ObjectIdentifier"/>
    <xs:element name="EncryptionInfoValue">
      <xs:complexType mixed="true">
        <xs:sequence>
          <xs:any minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ArchiveTimeStampSequenceType">
  <xs:sequence>
    <xs:element name="ArchiveTimeStampChain" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="DigestMethod" type="DigestMethodType"/>
          <xs:element name="CanonicalizationMethod" type="CanonicalizationMethodType"/>
          <xs:element name="ArchiveTimeStamp" type="ArchiveTimeStampType" maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="Order" type="OrderType" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ArchiveTimeStampType"/>
<xs:sequence>
  <xs:element name="HashTree" type="HashTreeType" minOccurs="0"/>
  <xs:element name="TimeStamp" type="TimeStampType"/>
  <xs:element name="Attributes" type="Attributes" minOccurs="0"/>
</xs:sequence>
<xs:attribute name="Order" type="OrderType" use="required"/>
</xs:complexType>

<xs:complexType name="DigestMethodType" mixed="true">
  <xs:sequence>
    <xs:element name="DigestToken" type="DigestTokenType" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="Algorithm" type="xs:anyURI" use="required"/>
</xs:complexType>

<xs:complexType name="CanonicalizationMethodType" mixed="true">
  <xs:sequence minOccurs="0">
    <xs:element name="CanonicalizationToken" type="CanonicalizationTokenType" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="Algorithm" type="xs:anyURI" use="required"/>
</xs:complexType>

<xs:complexType name="TimeStampType">
  <xs:sequence>
    <xs:element name="TimeStampToken">
      <xs:complexType mixed="true">
        <xs:complexContent mixed="true">
          <xs:restriction base="xs:anyType">
            <xs:sequence>
              <xs:element name="Type" type="xs:string" processContents="skip" minOccurs="0" maxOccurs="unbounded" use="required"/>
            </xs:sequence>
          </xs:restriction>
        </xs:complexContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:complexType>
</xs:element>

<xs:element name="CryptographicInformationList"
    type="CryptographicInformationType" minOccurs="0"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="HashTreeType">
    <xs:sequence>
        <xs:element name="Sequence" maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="DigestValue" type="xs:base64Binary"
                        maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="Order" type="OrderType"
                    use="required"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

<xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Attributes">
    <xs:sequence>
        <xs:element name="Attribute" maxOccurs="unbounded">
            <xs:complexType mixed="true">
                <xs:complexContent mixed="true">
                    <xs:restriction base="xs:anyType">
                        <xs:sequence>
                            <xs:any processContents="skip" minOccurs="0"
                                maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="Order" type="OrderType"
                            use="required"/>
                        <xs:attribute name="Type" type="xs:string"
                            use="optional"/>
                    </xs:restriction>
                </xs:complexContent>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="CryptographicInformationType">
  <xs:sequence>
    <xs:element name="CryptographicInformation"
                 maxOccurs="unbounded">
      <xs:complexType mixed="true">
        <xs:complexContent mixed="true">
          <xs:restriction base="xs:anyType">
            <xs:sequence>
              <xs:any processContents="skip" minOccurs="0"
                       maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="Order" type="OrderType"
                          use="required"/>
            <xs:attribute name="Type" type="xs:string"
                          use="required"/>
          </xs:restriction>
        </xs:complexContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="SupportingInformationType">
  <xs:sequence>
    <xs:element name="SupportingInformation"
                 maxOccurs="unbounded">
      <xs:complexType mixed="true">
        <xs:complexContent mixed="true">
          <xs:restriction base="xs:anyType"/>
        </xs:complexContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
8. Security Considerations

8.1. Secure Algorithms

Cryptographic algorithms and parameters that are used within Archive Time-Stamps must always be secure at the time of generation. This concerns the hash algorithm used in the hash lists of Archive Timestamp as well as hash algorithms and public key algorithms of the
timestamps. Publications regarding security suitability of cryptographic algorithms ([NIST.800-57-Part1.2006] and [ETSI TS 102 176-1 V2.0.0]) have to be considered during the verification. A generic solution for automatic interpretation of security suitability policies in electronic form is not the subject of this specification.

8.2. Redundancy

Evidence Records may become affected by weakening cryptographic algorithms even before this is publicly known. Retrospectively this has an impact on Archive Time-Stamps generated and renewed during the archival period. In this case the validity of Evidence Records created may end without any options for retroactive action.

Many TSAs are using the same cryptographic algorithms. While compromise of a private key of a TSA may compromise the security of only one TSA (and only on Archive Time-Stamp for example), weakening cryptographic algorithms used to generate Time-Stamp Tokens would affect many TSAs at the same time.

To manage such risks and to avoid the loss of Evidence Record validity due to weakening cryptographic algorithms used, it is RECOMMENDED to generate and manage at least two redundant Evidence Records for a single data object. In such scenarios redundant Evidence Records SHOULD use different hash algorithms within Archive Time-Stamp Sequences and different TSAs using different cryptographic algorithms for Time-Stamp Tokens.

8.3. Secure Time-Stamps

Archive Time-Stamps depend upon the security of normal Time-Stamping provided by TSA and stated in security policies. Renewed Archive Time-Stamps MUST have the same or higher quality as the initial Archive Time-Stamp of archive data. Archive Time-Stamps used for signed archive data SHOULD have the same or higher quality than the maximum quality of the signatures.
8.4. Time-Stamp verification

It is important to consider for renewal and verification that when a new Time-Stamp is applied, it MUST be ascertained that prior the time of renewal (i.e. when the new Time-Stamp is applied) the certificate of the before current Time-Stamp was not revoked due to a key compromise. Otherwise, in the case of a key compromise, there is the risk that the authenticity of the used Time-Stamp and therefore its security in the chain of evidence cannot be guaranteed. Other revocation reasons like the revocation for cessation of activity do not necessarily pose this risk as in that case the private key of the Time-Stamp unit would have been previously destroyed and thus cannot be used nor compromised.

Both elements <CryptographicInformationList> and <Attribute> are protected by future Archive Time_Stamp renewals and can store information as outlined in section 2.1. that is available at or before the time of the renewal of the specific Archive Time-Stamp. At the time of renewal all previous Archive Time-Stamp data structures become protected by the new Archive Time-Stamp and frozen by it, i.e. no data MUST be added or modified in these elements afterwards. If however, some supporting information is relevant for the overall Evidence Record or information that only becomes available later, this can be provided in the Evidence Record in the <SupportingInformationList> element. Data in the <SupportingInformatonList> can be added later to an Evidence Record, but it must rely on its own authenticity and integrity protection mechanism, like for example signed by current strong cryptographic means and/or provided by a trusted source (for example this could be the LTA providing its current system DSSC policy, signed with current strong cryptographic means).
9. IANA Considerations

This document defines the XML namespace "urn:ietf:params:xml:ns:ers" according to the guidelines in [RFC3688]. This namespace has been registered in the IANA XML Registry.

This document defines an XML schema (see Section 6) according to the guidelines in [RFC3688]. This XML schema has been registered in the IANA XML Registry and can be identified with the URN "urn:ietf:params:xml:schema:ers".

This specification defines a new IANA registry entitled "XML Evidence Record Syntax (ERSXML)". This registry contains two sub-registries entitled "Time-Stamp Token Type" and "Cryptographic Information Type". The policy for future assignments to both sub-registries is "RFC Required".

The sub-registry "Time-Stamp Token Type" contains textual names and description, which should refer to the specification or standard defining that type. It serves as assistance when validating a time-stamp token.

When registering a new Time-Stamp Token type, the following information MUST be provided:

- The textual name of the Time-Stamp Token type (value)
- A reference to a publicly available specification that defines the Time-Stamp Token type (description)

The initial values for the "Time-Stamp Token Type" sub-registry are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----</td>
<td>-----------</td>
<td>----------</td>
</tr>
</tbody>
</table>
The sub-registry "Cryptographic Information Type" contains textual names and description, which should refer to specification or standard defining that type. It serves as assistance when validating cryptographic information such as digital certificates, CRLs or OCSP-Responses.

When registering a new cryptographic information type, the following information MUST be provided:

- The textual name of the cryptographic information type (value)
- A reference to a publicly available specification that defines the cryptographic information type (description)

The initial values for the "Cryptographic Information Type" sub-registry are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT</td>
<td>DER-encoded X.509 Certificate</td>
<td>RFC 5280</td>
</tr>
<tr>
<td>CRL</td>
<td>DER-encoded X.509</td>
<td>RFC 5280</td>
</tr>
<tr>
<td></td>
<td>Certificate Revocation List</td>
<td></td>
</tr>
</tbody>
</table>
OCSP  DER-encoded OCSPResponse  RFC 2560
SCVP  DER-encoded SCVP response  RFC 5055

(CVResponse)

10. References

10.1. Normative References


10.2. Informative References


APPENDIX A: Detailed verification process of an Evidence Record

To verify the validity of an Evidence Record start with the first ATS till the last ATS (ordered by attribute Order) and perform verification for each ATS, as follows:

1. Select corresponding archive object and its data object or a group of data objects.

2. Re-encrypt data object or data object group, if <EncryptionInformation> field is used (see section 5. for more details)

3. Get a canonicalization method C and a digest method H from the <DigestMethod> element of the current chain.

4. Make a new list L of digest values of (binary representation of) objects (data, ATS or sequence) that MUST be protected with this ATS as follows:
   a. If this ATS is the first in the Archive Time-Stamp Chain:
      i. If this is the first ATS of the first ATSC (the initial ATS) in the ATSSeq, calculate digest values of data objects with H and add each digest value to the list L.
      ii. If this ATS is not the initial ATS, calculate a digest value with H of ordered ATSSeq without this and successive chains. Add value H and digest values of data objects to the list L.
   b. If this ATS is not the first in the ATSC:
      i. Calculate the digest value with H of the previous <TimeStamp> element and add this digest value to the list L.
5. Verify the ATS’s time-stamped value as follows. Get the first sequence of the hash tree for this ATS.

a. If this ATS has no hash tree elements then:

   ii. If this ATS is not the first in the ATSSeq (the initial ATS), then the time-stamped value must be equal to digest value of previous Time-Stamp element. If not, exit with a negative result.

   iii. If this ATS is the initial ATS in ATSC, there must be only one data object of the archive object. The digest value of that data object must be the same as its time-stamped value. If not, exit with a negative result.

b. If this ATS has a hash tree then: If there is a digest value in the list L of digest values of protected objects, which cannot be found in the first sequence of the hash tree or if there is a hash value in the first sequence of the hash tree which is not in the list L of digest values of protected objects, exit with a negative result.

   i. Get the hash tree from the current ATS and use H to calculate the root hash value (see sections 3.2.1. and 3.2.2.)

   ii. Get time-stamped value from the Time-Stamp Token. If calculated root hash value from the hash tree does not match the time-stamped value, exit with a negative result.

6. Verify Time-Stamp cryptographically and formally (validate the used certificate and its chain which may be available within the Time-Stamp Token itself or <CryptographicInformation> tag).
7. If this ATS is the last ATS, check formal validity for the current time (now), or get "valid from" time of the next ATS and verify formal validity at that specific time.

8. If the needed information to verify formal validity is not found within the Time-Stamp or within its Cryptographic Information section of ATS, exit with a negative result.

Author’s Addresses

Aleksej Jerman Blazic
SETCCE
Tehnoloski park 21
1000 Ljubljana
Slovenia

Phone: +386 (0) 1 620 4500
Fax:   +386 (0) 1 620 4509
Email: aljosa@setcce.si

Svetlana Saljic
SETCCE
Tehnoloski park 21
1000 Ljubljana
Slovenia

Phone: +386 (0) 1 620 4506
Fax:   +386 (0) 1 620 4509
Email: svetlana.saljic@setcce.si