Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

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.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on November 26, 2008.

Copyright Notice

Jerman Blazic, et. al. Expires November 6, 2008
Abstract

In many scenarios, users must be able to demonstrate the (time) existence, integrity and validity of data including signed data for long or undetermined period of time. This document specifies XML syntax and processing rules for creating evidence for long-term non-repudiation of existence of data. ERS-XML incorporates alternative syntax and processing rules to ASN.1 ERS 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 .................................................. 5
   1.1. Motivation ................................................. 5
   1.2. General Overview and Requirements ....................... 7
   1.3. Terminology ............................................... 8
   1.4. Conventions Used in This Document ...................... 10
2. Evidence Record ............................................... 10
   2.1. Evidence Record Structure .............................. 11
   2.2. Archive Timestamp Sequence and Archive Timestamp Chain Structure ........................................ 13
   2.2.1. Digest Method ....................................... 14
   2.3. Archive Timestamp Structure ................................
   2.3.1. Time-Stamp Token ..................................... 15
   2.3.2. Hash Tree .............................................. 16
   2.3.3. Merkle Hash-Tree ..................................... 18
     2.3.3.1. Generation of a Merkle Hash-Tree for a Group of Archive Objects ................................. 19
     2.3.3.2. Generation of the Reduced Hash Tree for an Archive Object ................................. 21
     2.3.3.3. Calculation of the root hash value from a reduced hash tree ........................................ 22
   2.3.4. Cryptographic Information ................................
3. Generation of an Evidence Record .............................. 23
   3.1. Initial Archive Timestamp ................................
   3.2. Renewal Process ...........................................
     3.2.1. Time-Stamp Renewal ..................................
     3.2.2. Hash Tree Renewal ..................................
4. Verification of an Evidence Record ............................ 28
5. Encryption ......................................................
6. XSD Schema for the Evidence Record ...........................
7. Security Considerations ....................................... 35
8. IANA Considerations ........................................... 35
9. Conclusions .................................................... 35
10. Acknowledgments ............................................... 35
1. Introduction

The purpose of the document is to define XML Schema and processing rules for Evidence Record Syntax in XML format. Document is related to initial ASN.1 syntax for Evidence Record Syntax.

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 information to prove data existence and integrity weakens or ceases to exist. Mechanisms such as digital signatures do not provide absolute reliability on a long term basis. Algorithms and cryptographic material used to create a signature can become weak in course of time and information needed to validate digital signatures may become compromised or simply cease to exist due to for example decomposing 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 related techniques used today suffer from the same problem of time related reliability degradation including techniques for time stamping, which are generally recognized as data existence proofs. Over long periods of time algorithms used may become weak or encryption keys compromised. Some of the problems might not even be technically related like decomposing time stamping authority. 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 [RFC3126]. Long term signature syntaxes address mostly the long term endurance of digital signatures, while evidence record syntax broadens this approach for data of any type or format.

The XMLERS 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 long term basis. XMLERS does not supplement the ERS syntax. It introduces the same approach but in a different format.

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.

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 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. The XML syntax supports archive data grouping (and de-grouping) together with simple or complex time stamp renewal process. Evidence records can be embedded in the data itself or stored separately as a standalone XML file.
1.2. General Overview and Requirements

ERSXML draft (draft-ietf-ltans-xmlers-02) specifies XML syntax and processing rules for creating evidence for long-term non-repudiation of existence of data in a unit called "Evidence Record". The XMLERS syntax is defined to meet the requirements for data structures as set out in [RFC4810]. This document also refers to 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. 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 out of the scope of this draft.

Evidence Record maintains a close relationship to time stamping techniques. However, timestamps 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 timestamps applied over a data object. Evidence for an archive object is created by acquiring a timestamp from a trustworthy authority for a specific value that is unambiguously related to a single or more data objects. The Evidence Record syntax enables processing of several archive objects within a single processing pass and by acquiring only one timestamp protects all archive objects.

Besides a timestamp other artifacts are also preserved in an 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 timestamp(s).

Due to the fact that digest algorithms or cryptographic methods used may become weak or that certificates used within a timestamp (and signed data) may be revoked or expired, the collected evidence data
must be monitored and renewed before such event occurs. Procedures for the generation and renewing of such evidences are already specified within the [RFC4998], but they depend on defined ASN.1 data structures. For the purpose of renewal of the evidence, digest values of ASN.1 formatted data must be calculated and used in further processing. Beside replacing an ASN.1 scheme with an XML scheme, this document also introduces XML based procedures and processing rules for the creation and 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 multitude of (archive) data objects, which for some reason (logically) belong together, e.g. a document file and a signature file could be an archive data object group, which represent signed data.

Archive Timestamp (ATS): An Archive Timestamp contains a time-stamp token, useful data for validation and potentially a list of hash values. The basic idea is to time-stamp a specific value, constructed from significant values (e.g. hash value of a list of hash values of documents), which are unambiguously related to the protected data objects.

Archive Timestamp Chain (ATSC): holds a sequence of Archive Timestamps generated during the preservation period.

Archive Timestamp Sequence (ATSSeq): is a sequence of Archive Timestamp Chains.

Canonicalization: Processing rules for transforming an XML document into its canonical form. Two XML document 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 list, etc.

Digest Method: Digest method is an identifier for 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.
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 one or more
given archived data objects over time. An evidence record includes
ordered collection of ATS’s, which are grouped into ATSC and ATSSeq.

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, i.e.
storage of archived data objects and evidence, etc.

Hash Tree: Collection of significant values of protected objects
(input objects and generated evidence within archival period). For
that purpose a Merkle Hash Tree [MER1980] may be constructed and
reduced for each archive object.

Reduced hash-tree: The process of reducing a Merkle hash-tree
[MER1980] to a list of lists of hash values. This is the basis of
storing the evidence for a single data object.
Timestamp (TS): A cryptographically secure confirmation generated by a Time Stamping Authority (TSA) [RFC3161] specifies a structure for timestamps and a protocol for communicating with a Timestamp 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].

According to the [RFC4998] specification an Archive Timestamp relates to a data object, if the hash value of this data object is part of the first hash value list of the Archive Timestamp. An Archive Timestamp relates to a data object group, if it relates to every data object of the group and no other data objects.

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, which is to be used to prove the existence of an archived object (a single data object or a group of data objects) at a certain time. It is possible to store Evidence Record separately from the archived object or to integrate it into the data itself.

Evidence Record Syntax enables processing of several archive objects (group processing) with a single process and by acquiring only one timestamp to protect many archive objects, without a need to access any other archive objects or their evidence records while demonstrating the validity for a particular archive object from the group.

The Evidence Record contains one or several Archive Time-Stamps (ATS). An ATS contains a time-stamp token and possibly useful data
for validation, like certificates, CRLs or OCSP 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 generated, which prolongs the validity of the archived object (of its data objects together with all previously generated archive time-stamps). This process must continue during the desired archiving period.

2.1. Evidence Record Structure

In XML syntax the Evidence Record is represented by the <EvidenceRecord> element, which has the following structure (where "+" denotes one or more occurrences and "*" denotes zero or more occurrences):

```xml
<EvidenceRecord>
  <Version />
  <EncryptionInformation>
    <EncryptionInformationType>
      <EncryptionInformationValue>
    </EncryptionInformationValue>
  </EncryptionInformation> *
  <ArchiveTimeStampSequence>
    <ArchiveTimeStampChain Order>
      <DigestMethod /> +
      <CanonicalizationMethod /> +
      <ArchiveTimeStamp Order>
        <HashTree /> *
        <TimeStamp /> +
        <CryptographicInformation /> *
        <Attributes /> *
      </ArchiveTimeStamp>) +
    </ArchiveTimeStampChain> +
  </ArchiveTimeStampSequence> +
</EvidenceRecord>
```

The XML tags have the following meanings:
<Version> tag 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 02.

<ArchiveTimeStampSequence> is a sequence of <ArchiveTimeStampChain>.

<CanonicalizationMethod> is a required element that specifies the canonicalization algorithm applied to the <ArchiveTimeStampSequence> or <ArchiveTimeStamp> element prior to performing digest value calculations.

<ArchiveTimeStampChain> holds a sequence of Archive Timestamps generated during the preservation period. Details on Archive Timestamp Chains and Archive Timestamp Sequences are described in section 2.3. The sequences of Archive Timestamp Chains and Archive Timestamps are ordered and the order must be indicated with "Order" attribute of the <ArchiveTimeStampChain> and <ArchiveTimeStamp> element.

<DigestMethod> tag identifies the digest algorithms used to calculate digest values over the archival period within an archive time-stamp chain from archive data object(s), archive time-stamps, archive time-stamp sequence and within time-stamp token.

<HashTree> tag holds a value or a structure of a reduced hash tree(s) described in section 2.3.2.

<TimeStamp> tag holds a time stamp token provided by the Time-Stamping Authority.

<CryptographicInformation> tag allows the storage of data needed in the process of archive time stamp token validation, when such data is not provided by the time stamp token itself (e.g. time stamp in XML format). This could include possible Trust Anchors,
certificates, revocation information or the current definition of the suitability of cryptographic algorithms, past and present. These items may be added based on the policy used. This data is protected by successive time-stamps in the sequence of the time-stamps.

<Attributes> tag contains additional information that may be provided by an LTA used for the renewal process. An example of additional information may be processing (renewal) policies, which are relevant for document(s) preservation and evidence validation at a later stage.

<EncryptionInformation> tag holds information on cryptographic algorithms and material used to encrypt archive data. This optional information is needed to unambiguously re-encrypt data objects. When omitted, data objects are not encrypted or non-repudiation proof is not needed for the unencrypted data.

2.2. Archive Timestamp Sequence and Archive Timestamp Chain Structure

<ArchiveTimeStampSequence> element contains an ordered sequence of <ArchiveTimeStampChain> elements and <ArchiveTimeStampChain> element an ordered sequence of <ArchiveTimeStamp> elements. Order is indicated with the Order attribute.

The first <ArchiveTimeStampChain> element and its first <ArchiveTimeStamp> are generated at the beginning of the archival period, both having values of the Order attribute equal 1.

When this initial <ArchiveTimeStamp> must be renewed, a new <ArchiveTimeStamp> is generated and depending on the generation process, it is either placed:

- as <ArchiveTimeStamp> element in the same <ArchiveTimeStampChain> element (see Time-Stamp Renewal) or
as the first element in the sequence (see Hash Tree Renewal) in a newly created <ArchiveTimeS

The value of its Order attribute is increased by one, and also if a new chain is created its Order attribute is increased by one relative to the one in the previous chain.

Generally when a new ATS is created it is placed into the last ATSC as the last child element with an increased Order value of 1 from the preceding ATS or a new ATSC is created with increased Order value of 1 from the preceding ATSC element and the new ATS with the Order number 1 is placed as its first child element.

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.

2.2.1. Digest Method

Digest method is a required element that identifies the digest algorithm used to digest the archive data object(s) and the previously generated long-term evidence over the archival period. It is specified at the level of the ATSC and indicates the digest algorithm that MUST be used for all digest value calculations related to the archive timestamps within this chain.

Digest algorithms used for evidence record correspond to the algorithms used for time stamp token(s) within a single ATSC. When algorithms used by TSA are changed (e.g. upgraded) a new ATSC must be started using new digest algorithm. New hash values used for new ATSC must be obtained both, for archive data object(s) and for previous ATSCs (see detailed description in section 3.2.).

Within an archive time-stamp chain, depending on the case involved, following digest value(s) are calculated:
1. digest value(s) of archived data object(s) at the beginning of the archival period or when performing hash-tree renewal

2. digest value(s) of previous ATS(s) when performing timestamp renewal

3. digest value(s) of previous ATSC(s) when performing hash tree renewal

For cases 2) and 3) calculation of the digest value of an XML element is needed. Before performing digest value calculation 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.

2.3. Archive Timestamp Structure

The process of construction of an ATS must unambiguously bind the archived object and the time-stamped value and thus prove that the archived object existed and was identical, at the time of the timestamp, to the currently present archived object (at the time of verification).

Therefore an ATS is a collection of the time-stamp token, an optional structure (a hash tree) for digest values of objects that were protected with that time-stamp token and optional structures (cryptographic information) that store additional data needed for formal verification of the time-stamp token, such as certificate chain or certificate revocation list.

For the initial ATS the value to be time-stamped must be unambiguously related to the archived object (to all of its data objects). When the same digest algorithm is used with the successive
time-stamps (in the renewal process), it is enough that the time-stamped value is related only to the digest value of the last (previous) ATS. When a different digest algorithm is used in the renewal process, the time-stamped value must be unambiguously related to the archive object and all previously created ATSCs. Renewal process is described in detail in section 3.2.

2.3.1. Time-Stamp Token

Time-Stamp is an attestation generated by a TSA that a data item existed at a certain time. For example, [RFC3161] specifies a structure for signed time-stamp tokens in ASN.1 format. The following structure example (reference to the Entrust XML Schema for time-stamp) 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.

```xml
<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>
```

2.3.2. Hash Tree

The time-stamping service may be, for a large number of archived objects, expensive and time-demanding, so the LTA may profit from
acquiring one time-stamp for many archived objects, which are not otherwise related to each other. For that purpose a Merkle Hash Tree [MER1980] may be constructed and reduced for each archive object.

Hash tree structure is a container for significant values, needed to unambiguously relate a time-stamped value to protected data objects, and is represented by the <HashTree> element.

The lists of digest values are generated by reduction of an ordered Merkle hash-tree. The leaves of this hash-tree are the digest values of the data objects to be time-stamped. Every inner node of the tree contains one digest value, which is generated by digesting the binary sorted concatenation of the children nodes (leaves). The root digest value, which represents unambiguously all data objects, is time-stamped.

Note that there are no restrictions to the quantity of hash value lists and of their length. Also note that it is profitable but not required to build hash-trees and reduce them. An Archive Time-Stamp may consist only of one list of hash-values and a time-stamp or in the extreme case, only a time-stamp with no hash value lists.

A sample of lists of hash values within the Content node value:

```xml
<Sequence>
  <DigestValue>5XQCAGgwJL2WZ6nv20SGY1RFpK8=</DigestValue>
  <DigestValue>wyFrW58ATzRch7VUPNY2P+75Q/I=</DigestValue>
  <DigestValue>woCvx62tw0uc24v51xtNNxld5Kw=</DigestValue>
  <DigestValue>7N21phOrGx+/PCxtGwzbKIj+InQ=</DigestValue>
  <DigestValue>zbCNhmQv+8kRo9W/0YedrpeZ1a8=</DigestValue>
  <DigestValue>ThuhSOZhnj42vsIRg38epxj9qVo=</DigestValue>
</Sequence>
<Sequence>
  <DigestValue>FX7AgSsZ0kaW8fHWi4BYDkAkZS0=</DigestValue>
</Sequence>
```
This sample represents a reduced hash tree. The first sequence (input list) contains 6 digest values, which indicate 6 hash values of data objects to be archived with archive time-stamp element. Sequences that follow the input list are used to calculate the final digest value to be time-stamped. The reduced tree is result of reducing a Merkle hash-tree as described in chapter 2.3.3. The digest values are always represented as encoded base64 character data.

2.3.3. Merkle Hash-Tree

A hash tree is a tree of digest values in which the leaves are hashes of data blocks in, for instance, a file or set of files. Nodes further up in the tree are the hashes of their respective children. At the top of a hash tree is the root hash value.

From Merkle Hash-Tree for each leaf a very small sub tree may be extracted (a reduced tree), that holds enough information to unambiguously bind the leaf value with the root hash value. After the group processing of several archive objects (ie the calculation of the root hash value, that unambiguously binds together all archive objects, was time-stamped), for each archive object a reduced tree is saved within its evidence record (ie a hash tree within the last ATS).
2.3.3.1. Generation of a Merkle Hash-Tree for a Group of Archive Objects

The Merkle Hash-Tree for a group of archive objects is built from bottom to the root. First are collected the leaves of the tree. The leaves are digest values of archive objects:

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

2. Calculate hash values of the archive objects and put them into input list 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 some hash values on the input list (archive objects having more than one data object) also lists of their sub-hash values are stored.

3. Group together items in the input list by N (to make binary tree in pairs) and for each group: binary ascending sort, concatenate and calculate hash values with algorithm H. The result is a new input list.

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

Example: The input list with 18 hashes, 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 having the same number of children.
2.3.3.2. Generation of the Reduced Hash Tree for an Archive Object

The following procedure describes generation of the reduced hash tree for an archive object:

1. For a selected archive object generate the first sequence of the reduced tree, which contains the list of hash values of the data objects contained in the archive object (one or more). Select the node with the hash value of the archive object.

2. Select all neighboring nodes, which have the same parent as currently selected node and add their hash values as a new sequence to the reduced tree. Select its parent node.

3. Repeat step 2 until the root is reached. Note that parent nodes are not saved in the list as they are computable.

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

```xml
<Sequence>
  <DigestValue>h4</DigestValue>
  <DigestValue>h5</DigestValue>
  <DigestValue>h6</DigestValue>
  <DigestValue>h7</DigestValue>
</Sequence>
<Sequence>
  <DigestValue>h8</DigestValue>
  <DigestValue>h9</DigestValue>
</Sequence>
<Sequence>
  <DigestValue>h’1</DigestValue>
</Sequence>
```

Figure 1 Generation of the Reduced Hash Tree.
<DigestValue>h''3</DigestValue>
</Sequence>
<Sequence>
  <DigestValue>h''2</DigestValue>
</Sequence>

### 2.3.3.3. Calculation of the root hash value from a reduced hash tree

The following procedure describes generation of the root hash value from a reduced hash tree:

1. Take the first sequence from a reduced hash tree, sort hash items, concatenate them and calculate a hash value with algorithm H (the one used for creation of the hash tree).

2. Remove this sequence from the reduced hash tree.

3. If a reduced hash tree is not empty, add previously calculated hash value to the first sequence and go to step 1. If a reduced hash tree is empty, then the last calculated hash value is the root hash value.

### 2.3.4. Cryptographic Information

Digital certificates, CRLs 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 `<CryptographicInformation>` element (as a node value of its `<Content>` element).

The attribute Type is optional and is used to store processing information about type of stored cryptographic information.
3. Generation of an Evidence Record

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

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

2. Create the initial ATS. This is the first ATS within the initial Archive Time-Stamp Chain of the Archive Time-Stamp Sequence.

3. Refresh the Archive Time-Stamp when necessary, by Time-Stamp Renewal or Hash-Tree Renewal.

In the case that only essential parts of documents or objects shall be protected, the application not defined in this draft must ensure that the correct extraction of binary data is made for generation of evidence record.

For example: an application may provide also evidence such as certificates, revocation lists etc., needed to verify and validate signed data objects. This evidence may be added to the archived group of data object and will be protected within initial (and successive) time-stamp(s).

Comment [AJB]: For the purpose of extraction, inclusion of reference tag should be defined, i.e. information on which data or piece of data is digested.

3.1. Initial Archive Timestamp

Initial ATS relates to a data object or a data object group that represents an archived object. The generation of the initial ATS elements can be done in a single process pass for one or for many archived objects, described as follows:
1. Collect one or more archived objects to be time-stamped.

2. Select a valid digest algorithm H. The same digest algorithm MUST be used to create the time-stamped values and the time-stamp.

3. Create an input list of digest values of archive objects calculated with H (one digest value for each archived data object). Those digest values are the leaves of the hash tree for the whole group of archived objects.

A hash tree to be included within the initial ATS of a single archived object is generated as a reduced hash tree from the hash tree for the whole group as defined in 2.3.3. Hash tree may be omitted in the initial archive time-stamp, when an archive object is having a single data object; then the time-stamped value must match the digest value of that single data object. When an archive object is composed of more than one data object, place digest values of all data objects in a group, sort them in binary ascending order, concatenate them into a single string and generate a new digest value. This digest value represents a digest value of that archived object. The list of digest values of data objects must be part of the first sequence in the hash tree (reduced hash tree) for this archive object.

4. If there is more than one digest value on the input list, place them in groups and sort each group in binary ascending order. Concatenate group digest values and generate new digest values, which represent inner nodes of the hash tree for the whole group. Repeat this step until there is only one hash value left, which is the root node value of the hash tree.

5. Acquire time-stamp for root node value. If the time-stamp is valid, the initial archive time-stamp may be generated.
3.2. Renewal Process

Before the cryptographic algorithms used within the last Archive Time-Stamp become weak or the time-stamp certificates are invalidated, the existence of Archive Time-Stamp or archive time-stamped data has to be reassured. This can be done by creating a new archive time-stamp. Depending on whether the time-stamp becomes invalid or the hash algorithm of the hash tree becomes weak, two kinds of renewal processes are possible.

If the digest algorithm to be used (H) in the renewal process is the same as one used in the last Archive Time-Stamp (H’), the digest value of that Archive Time-Stamp is calculated and a new Archive Time-Stamp is applied. This process is known simply as time-stamp renewal.

The process of hash tree renewal occurs when the new digest algorithm is (due to new cryptographic constrains) different than the one used in the last Archive Time-Stamp (H <> H’). In this case new Archive Time-stamp Chain is created and a digest value for a new Archive Time-Stamp is digested in the following manner from the binary ascending sorted and concatenated:

- digest values of data object(s) calculated with the new digest algorithm and
- digest value of all previously created Archive Time-Stamp Chains calculated with the new digest algorithm (including ordered archive timestamp chains and contained archive time stamps with hash trees, cryptographic information, etc.).

3.2.1. Time-Stamp Renewal

For the purpose of Time-Stamp Renewal, the complete content of the <ArchiveTimeStamp> element of the preceding Archive Timestamp MUST be
hashed and time stamped by a new Archive Time-Stamp. A digest value to be time-stamped within a new ATS is calculated as follows:

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

2. Digest value is calculated from binary representation of the last <ArchiveTimeStamp> element including added cryptographic information. Acquire the time-stamp for the calculated digest 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 Archive Time-Stamp Chain.

The new ATS and its hash tree MUST use the same digest algorithm as the preceding one, which is specified in the <DigestMethod> element of the ATSC.

3.2.2. Hash Tree Renewal

Hash tree renewal process is performed in cases where the used hash algorithm becomes weak. This process takes into account values of all preceding Archive Time-Stamps as well as values of archive data objects (covered by Archive Time-Stamps). Hash tree renewal procedure is as follows:
1. If the current ATS does not contain needed proof for long-term formal validation of its time-stamp token within the time-stamp token, collect needed data such as root certificates, certificate revocation lists, etc., and include them in <CryptographicInformation> element of the last Archive Time-Stamp (each data object into a separate <CryptographicInformation>).

2. Select a (new) secure hash algorithm H and select data objects d(i) referred to by initial Archive Time-Stamp. Generate hash values h(i) = H(d(i)). In case the initial Archive Time-Stamp is applied to more than one data object (of archive data object), then more than one hash values are generated i.e., h(i_a), h(i_b), .., h(i_n)

3. Calculate the digest value hatsc(i) = H(ATSC(i)) of the canonicalized binary representation of the previously generated and ordered <ArchiveTimeStampChain> elements within <ArchiveTimeStampsSequence> element, corresponding to data object d(i). Note that Archive Timestamp Chains and Archive Time Stamps MUST be chronologically ordered, each respectively to its Order attribute.

4. Concatenate and sort in binary ascending order: each h(i) and corresponding H(ATSC(i)) and generate a new digest value h(i)’. In case of more data objects in one archive data object, concatenate and sort in binary ascending order hash values h(i_a), h(i_b), etc. and H(ATSC(i)). Build a new Archive Timestamp for each h(i)’. Generation of the reduced hash tree (for h(i)’ elements) is defined in section 2.3.2. Note that each h(i)’ is treated as the document hash in section 2.3.2. The first hash value list in the reduced hash tree should only contain h(i)’. For a multi-document group (data object group which contains more than one document), the first hash value list contains the new hashes for all the documents in this group in binary ascending order, i.e. h(i_a)’, h(i_b)’, etc.
5. Create new `<ArchiveTimeStampChain>`, and place it into the existing `<ArchiveTimeStampSequence>` as a last child with the order number increased by one. Create new `<ArchiveTimeStamp>` element with order number 1 and place it into created `<ArchiveTimeStampChain>` element.

4. Verification of an Evidence Record

An Evidence Record shall prove that an archive object existed and has not been changed from the time of the time-stamp token within the first ATS. Every ATS, but the last, must be valid at the time of the next ATS. In order to complete the non-repudiation proof for the data objects, the last ATS has to be valid at the time when verification is performed.

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 an archive data object or 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 digest method identifier $H$ from the `<DigestMethod>` element of the current ATSC.

4. Make a list of digest values of (archive) data objects within (archive) data object group that MUST be protected with this ATS as follows:

   a. If this ATS is the first in the ATSC chain:
i. If this is the first ATS of the first ATSC in the ATSSeq sequence, calculate digest values of data objects with H and add each digest value to the list.

ii. If this is the first ATS of the ATSC which is not the initial ATSC in the ATSSeq sequence, calculate a single digest value with H of ordered ATSCs. Add and sort in binary ascending order this digest value with digest values of protected data objects and generate a new hash value.

b. If this ATS is not the first in the ATSC chain:

i. Calculate the digest value with H of the previous ATS element.

5. Get the first sequence of the hash tree for this ATS. If this ATS has no hash tree elements then:

a. If this ATS is not the first in the ATSSeq (first ATS of first ATSC), exit with a negative result.

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

6. If there is a digest value in the list of digest values of protected objects, which can not 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 of digest values of protected objects, exit with a negative result. Get the hash tree from the current ATS and use H to calculate the root hash value (see section 2.3.3.3.)
7. 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.

8. Verify timestamp cryptographically and formally (validate the used certificate and its chain which may be available within the time stamp token itself or <CryptographicInformation> tag).

9. 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.

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

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 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 however 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 have to be properly preserved otherwise the evidence record is not a non-repudiation proof. To achieve this, cryptographic material 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 LTA for preservation is kept in an unencrypted form by a client. In such scenarios cryptographic material is used to re-encrypt unencrypted data for the purpose of performing validation of evidence record, which is related to the encrypted form of client’s data.

[AJB] This section may need rework (detailed definition of cryptographic information element values).

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, (while the element holds e.g. encryption key value).

6. XSD Schema for the Evidence Record

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.setcce.org/schemas/ers"
targetNamespace="http://www.setcce.org/schemas/ers" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="EvidenceRecord">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="EncryptionInformation" type="EncryptionInfo" minOccurs="0"/>
        <xs:element name="ArchiveTimeStampSequence">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="EncryptionInformation" type="EncryptionInfo"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
<xs:sequence>
  <xs:element name="ArchiveTimeStampChain"
    minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ArchiveTimeStamp" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="HashTree" type="HashTreeType" minOccurs="0"/>
              <xs:element name="TimeStamp">
                <xs:complexType mixed="true">
                  <xs:sequence>
                    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="CryptographicInformation" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType mixed="true">
                  <xs:sequence>
                    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>
<xs:element name="Attributes" type="Attributes" minOccurs="0" name="Order" type="xs:integer" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="CanonicalizationMethod" type="CanonicalizationMethodType"/>
<xs:element name="DigestMethod" type="DigestMethodType"/>
<xs:attribute name="Order" type="xs:integer" use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<!-- TYPE DEFINITIONS-->
<xs:complexType name="EncryptionInfo">
<xs:sequence>
<xs:element name="EncryptionInfoType" type="ObjectIdentifier"/>
<xs:element name="EncryptionInfoValue" type="OpenType"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="DigestMethodType" mixed="true">
<xs:sequence>
<xs:any namespace="##other" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:attribute name="Algorithm" type="xs:anyURI" use="required"/>
</xs:complexType>
<xs:complexType name="CanonicalizationMethodType" mixed="true">
  <xs:sequence>
    <xs:any namespace="##any" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="Algorithm" type="xs:anyURI" use="required"/>
</xs:complexType>

<xs:complexType name="HashTreeType" mixed="true">
  <xs:sequence>
    <xs:element name="Sequence" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="DigestValue" type="xs:base64Binary" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Attributes">
  <xs:sequence>
    <xs:element name="Attribute" type="Attribute"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Attribute">
  <xs:sequence>
    <xs:element name="attrType" type="ObjectIdentifier"/>
    <xs:element name="attrValues">
      <xs:complexType>
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
          <xs:element name="AttributeValue" type="AttributeValue"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="AttributeValue"/>
</xs:complexType>
</xs:complexType>
<xs:any processContents="lax"/>
</xs:sequence>
</xs:complexType>
<xs:simpleType name="ObjectIdentifier">
    <xs:restriction base="xs:token">
        <xs:pattern value="[0-2](\.[1-3]?[0-9]?(\d+))?"/>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="OpenType">
    <xs:sequence>
        <xs:any processContents="lax"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>

7. Security Considerations
TBA

8. IANA Considerations
TBA

9. Conclusions
TBA

10. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.
APPENDIX A: First Appendix

TBA
11. References

[I-D.ietf-ltans-ers] Brandner, R., "Evidence Record Syntax (ERS)",
    draft-ietf-ltans-ers-11 (work in progress), February 2007

    (LTAP)", draft-ietf-ltans-ltap-03 (work in progress),
    October 2006.

    Requirements", draft-ietf-ltans-reqs-10 (work in progress),
    December 2006.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zucherato,
    "Internet X.509 Public Key Infrastructure Time-Stamp

[RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet
    X.509 Public Key Infrastructure Certificate and Certificate

[RFC3852] Housley, R., "Cryptographic Message Syntax (CMS)", RFC
    3852, July 2004.


/XMLDsig] Eastlake, D., "XML-Signature Syntax and
    Processing", XMLDsig, July 2006.
11.1. Normative References

TBA

11.2. Informative References


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

Tobias Gondrom  
Waisenhausstr. 67C  
80637 Munich  
Germany  

Phone: +49 (0) 89 3205 330  
Fax: /  
Email: tobias.gondrom@gondrom.org

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this
specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The IETF Trust (2008).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.