Network Working Group   T. Gondrom
Request for Comments: 4998   Open Text Corporation
Category: Standards Track   R. Brandner
                                               InterComponentWare AG
                                                        U. Pordesch
                                             Fraunhofer Gesellschaft
                                                        August 2007

                    Evidence Record Syntax (ERS)

Status of This Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

   In many scenarios, users must be able prove the existence and
   integrity of data, including digitally signed data, in a common and
   reproducible way over a long and possibly undetermined period of
   time. This document specifies the syntax and processing of an
   Evidence Record, a structure designed to support long-term non-
   repudiation of existence of data.

Table of Contents

1. Introduction

1.1. Motivation

   In many application areas of electronic data exchange, a non-
   repudiable proof of the existence of digital data must be possible.
   In some cases, this proof must survive the passage of long periods of
   time. An important example is digitally signed data. Digital
   signatures can be used to demonstrate data integrity and to perform
   source authentication. In some cases, digitally signed data must be
   archived for 30 years or more. However, the reliability of digital
   signatures over long periods is not absolute. During the archival
   period, hash algorithms and public key algorithms can become weak or
   certificates can become invalid. These events complicate the
   reliance on digitally signed data after many years by increasing the
   likelihood that forgeries can be created. To avoid losing the
   desired security properties derived from digital signatures, it is
   necessary to prove that the digitally signed data already existed
   before such a critical event. This can be accomplished using a
   timestamp. However, some timestamps rely upon mechanisms that will
   be subject to the same problems. To counter this problem, timestamps
   are renewed by simply obtaining a new timestamp that covers the
   original data and its timestamps prior to the compromise of
   mechanisms used to generate the timestamps. This document provides a
   syntax to support the periodic renewal of timestamps.

   It is necessary to standardize the data formats and processing
   procedures for such timestamps in order to be able to verify and
   communicate preservation evidence. A first approach was made by IETF
   within [RFC3126], where an optional Archive Timestamp Attribute was
   specified for integration in signatures according to the
   Cryptographic Messages Syntax (CMS) [RFC3852].

   Evidence Record Syntax (ERS) broadens and generalizes this approach
   for data of any format and takes long-term archive service
   requirements [RFC4810] into account -- in particular, the handling of
   large sets of data objects. ERS specifies a syntax for an
   EvidenceRecord, which contains a set of Archive Timestamps and some
   additional data. This Evidence Record can be stored separately from
   the archived data, as a file, or integrated into the archived data,
   i.e., as an attribute. ERS also specifies processes for generation
   and verification of Evidence Records. Appendix A describes the
   integration and use of an EvidenceRecord in context of signed and
   enveloped messages according to the Cryptographic Message Syntax
   (CMS). ERS does not specify a protocol for interacting with a long-
   term archive system. The Long-term Archive Protocol specification
   being developed by the IETF LTANS WG addresses this interface.

1.2. General Overview and Requirements

   ERS is designed to meet the requirements for data structures set
   forth in [RFC4810].

   The basis of the ERS are Archive Timestamps, which can cover a single
   data object (as an RFC3161 compliant timestamp does) or can cover a
   group of data objects. Groups of data objects are addressed using
   hash trees, first described by Merkle [MER1980], combined with a
   timestamp. The leaves of the hash tree are hash values of the data
   objects in a group. A timestamp is requested only for the root hash
   of the hash tree. The deletion of a data object in the tree does not
   influence the provability of others. For any particular data object,
   the hash tree can be reduced to a few sets of hash values, which are
   sufficient to prove the existence of a single data object.
   Similarly, the hash tree can be reduced to prove existence of a data
   group, provided all members of the data group have the same parent
   node in the hash tree. Archive Timestamps are comprised of an
   optional reduced hash tree and a timestamp.

   An EvidenceRecord may contain many Archive Timestamps. For the
   generation of the initial Archive Timestamp, the data objects to be
   timestamped have to be determined. Depending on the context, this
   could be a file or a data object group consisting of multiple files,
   such as a document and its associated digital signature.

   Before the cryptographic algorithms used within the Archive Timestamp
   become weak or timestamp certificates become invalid, Archive
   Timestamps have to be renewed by generating a new Archive Timestamp.
   (Note: Information about the weakening of the security properties of
   public key and hash algorithms, as well as the risk of compromise of
   private keys of Time Stamping Units, has to be closely watched by the
   Long-Term Archive provider or the owner of the data objects himself.
   This information should be gathered by "out-of-band" means and is out
   of scope of this document.) ERS distinguishes two ways for renewal
   of an Archive Timestamp: Timestamp Renewal and Hash-Tree Renewal.

   Depending on the conditions, the respective type of renewal is
   required: The timestamp renewal is necessary if the private key of a
   Timestamping Unit has been compromised, or if an asymmetric algorithm
   or a hash algorithm used for the generation of the timestamps is no
   longer secure for the given key size. If the hash algorithm used to
   build the hash trees in the Archive Timestamp loses its security
   properties, the Hash-Tree Renewal is required.

   In the case of Timestamp Renewal, the timestamp of an Archive
   Timestamp has to be hashed and timestamped by a new Archive
   Timestamp. This mode of renewal can only be used when it is not

necessary to access the archived data objects covered by the
timestamp. For example, this simple form of renewal is sufficient if
the public key algorithm of the timestamp is going to lose its
security or the timestamp authority certificate is about to expire.
This is very efficient, in particular, if Archive Timestamping is
done by an archiving system or service, which implements a central
management of Archive Timestamps.

Timestamp renewal is not sufficient if the hash algorithm used to
build the hash tree of an Archive Timestamp becomes insecure. In the
case of Hash-Tree Renewal, all evidence data must be accessed and
timestamped. This includes not only the timestamps but also the
complete Archive Timestamps and the archived data objects covered by
the timestamps, which must be hashed and timestamped again by a new
Archive Timestamp.

1.3. Terminology

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

Archived data object group: A set of two or more of data objects,
which for some reason belong together. For example, a document file
and a signature file could be an archived data object group, which
represent signed data.

Archive Timestamp: A timestamp and typically lists of hash values,
which allow the verification of the existence of several data objects
at a certain time. (In its most simple variant, when it covers only
one object, it may only consist of the timestamp.)

Archive Timestamp Chain: Part of an Archive Timestamp Sequence, it is
a time-ordered sequence of Archive Timestamps, where each Archive
Timestamp preserves non-repudiation of the previous Archive
Timestamp, even after the previous Archive Timestamp becomes invalid.
Overall non-repudiation is maintained until the new Archive Timestamp
itself becomes invalid. The process of generating such an Archive
Timestamp Chain is called Timestamp Renewal.

Archive Timestamp Sequence: Part of the Evidence Record, it is a
sequence of Archive Timestamp Chains, where each Archive Timestamp
Chain preserves non-repudiation of the previous Archive Timestamp
Chains, even after the hash algorithm used within the previous
Archive Timestamp's hash tree became weak. Non-repudiation is
preserved until the last Archive Timestamp of the last chain becomes
invalid. The process of generating such an Archive Timestamp
Sequence is called Hash-Tree Renewal.

Evidence: Information that may be used to resolve a dispute about
various aspects of authenticity of archived data objects.

Evidence record: Collection of evidence compiled for one or more
given archived data objects over time. An evidence record includes
all Archive Timestamps (within structures of Archive Timestamp Chains
and Archive Timestamp Sequences) and additional verification data,
like certificates, revocation information, trust anchors, policy
details, role information, etc.

Long-term Archive (LTA) Service: A service responsible for preserving
data for long periods of time, including generation and collection of
evidence, storage of archived data objects and evidence, etc.

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

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. An Archive Timestamp Chain relates to a data object / data
object group, if its first Archive Timestamp relates to this data
object/data object group. An Archive Timestamp Sequence relates to a
data object / data object group, if its first Archive Timestamp Chain
relates to this data object/data object group.

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. Identification and References

2.1. ASN.1 Module Definition

   As many open ASN.1 compilers still support the 1988 syntax, this
   standard offers to support two versions of ASN.1 1997-ASN.1 and 1988-
   ASN.1. (For specification of ASN.1 refer to [CCITT.X208.1988],
   [CCITT.X209.1988], [CCITT.X680.2002] and [CCITT.X690.2002].) This
   specification defines the two ASN.1 modules, one for 1988 conform
   ASN.1 and another in 1997-ASN.1 syntax. Depending on the syntax
   version of your compiler implementation, you can use the imports for
   the 1988 conformant ASN.1 syntax or the imports for the 1997-ASN.1
   syntax. The appendix of this document lists the two complete
   alternative ASN.1 modules. If there is a conflict between both
   modules, the 1988-ASN.1 module precedes.

2.1.1. ASN.1 Module Definition for 1988 ASN.1 Syntax

   1988 ASN.1 Module start

   ERS {iso(1) identified-organization(3) dod(6)
        internet(1) security(5) mechanisms(5)
        ltans(11) id-mod(0) id-mod-ers88(2) id-mod-ers88-v1(1) }
   DEFINITIONS IMPLICIT TAGS ::=
   BEGIN

2.1.2. ASN.1 Module Definition for 1997-ASN.1 Syntax

   ASN.1 Module start

   ERS {iso(1) identified-organization(3) dod(6)
        internet(1) security(5) mechanisms(5)
        ltans(11) id-mod(0) id-mod-ers(1) id-mod-ers-v1(1) }
   DEFINITIONS IMPLICIT TAGS ::=
   BEGIN

2.2. ASN.1 Imports and Exports

   The specification exports all definitions and imports various
   definitions. Depending on the ASN.1 syntax version of your
   implementation, you can use the imports for the 1988 conform ASN.1
   syntax below or the imports for the 1997-ASN.1 syntax in
   Section 2.2.2.

2.2.1. Imports and Exports Conform with 1988 ASN.1

   -- EXPORTS ALL --

   IMPORTS

    -- Imports from RFC 3852 Cryptographic Message Syntax
   ContentInfo, Attribute
       FROM CryptographicMessageSyntax2004 -- FROM [RFC3852]
         { iso(1) member-body(2) us(840) rsadsi(113549)
           pkcs(1) pkcs-9(9) smime(16) modules(0) cms-2004(24) }

     -- Imports from RFC 3280 [RFC3280], Appendix A.1
   AlgorithmIdentifier
       FROM PKIX1Explicit88
           { iso(1) identified-organization(3) dod(6)
           internet(1) security(5) mechanisms(5) pkix(7)
           mod(0) pkix1-explicit(18) }
   ;

2.2.2. Imports and Exports Conform with 1997-ASN.1

   -- EXPORTS ALL --

   IMPORTS

    -- Imports from PKCS-7
   ContentInfo
       FROM PKCS7
           {iso(1) member-body(2) us(840) rsadsi(113549)
           pkcs(1) pkcs-7(7) modules(0)}

     -- Imports from AuthenticationFramework
   AlgorithmIdentifier
       FROM AuthenticationFramework
           {joint-iso-itu-t ds(5) module(1)
           authenticationFramework(7) 4}

    -- Imports from InformationFramework
   Attribute
       FROM InformationFramework
           {joint-iso-itu-t ds(5) module(1)
           informationFramework(1) 4}
   ;

2.3. LTANS Identification

   This document defines the LTANS object identifier tree root.

   LTANS Object Identifier tree root

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

3. Evidence Record

   An Evidence Record is a unit of data, which can be used to prove the
   existence of an archived data object or an archived data object group
   at a certain time. The Evidence Record contains Archive Timestamps,
   generated during a long archival period and possibly useful data for
   validation. It is possible to store this Evidence Record separately
   from the archived data objects or to integrate it into the data
   itself. For data types, signed data and enveloped data of the CMS
   integration are specified in Appendix A.

3.1. Syntax

   Evidence Record has the following ASN.1 Syntax:

   ASN.1 Evidence Record

   EvidenceRecord ::= SEQUENCE {
       version   INTEGER { v1(1) } ,
       digestAlgorithms   SEQUENCE OF AlgorithmIdentifier,
       cryptoInfos   [0] CryptoInfos OPTIONAL,
       encryptionInfo   [1] EncryptionInfo OPTIONAL,
       archiveTimeStampSequence ArchiveTimeStampSequence
       }

   CryptoInfos ::= SEQUENCE SIZE (1..MAX) OF Attribute

   The fields have the following meanings:

   The 'version' field 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 ERS. The
   value 1 indicates this specification. Lower values indicate an
   earlier version of the ERS has been used. An implementation
   conforming to this specification SHOULD reject a version value below
   1.

digestAlgorithms is a sequence of all the hash algorithms used to
hash the data object over the archival period. It is the union of
all digestAlgorithm values from the ArchiveTimestamps contained in
the EvidenceRecord. The ordering of the values is not relevant.

cryptoInfos allows the storage of data useful in the validation of
the archiveTimeStampSequence. This could include possible Trust
Anchors, certificates, revocation information, or the current
definition of the suitability of cryptographic algorithms, past and
present (e.g., RSA 768-bit valid until 1998, RSA 1024-bit valid until
2008, SHA1 valid until 2010). These items may be added based on the
policy used. Since this data is not protected within any timestamp,
the data should be verifiable through other mechanisms. Such
verification is out of scope of this document.

encryptionInfo contains the necessary information to support
encrypted content to be handled. For discussion of syntax, please
refer to Section 6.1.

ArchiveTimeStampSequence is a sequence of ArchiveTimeStampChain,
described in Section 5.

If the archive data objects were encrypted before generating Archive
Timestamps but a non-repudiation proof is needed for unencrypted data
objects, the optional encryptionInfos field contains data necessary
to unambiguously re-encrypt data objects. If omitted, it means that
data objects are not encrypted or that a non-repudiation proof for
the unencrypted data is not required. For further details, see
Section 6.

3.2. Generation

The generation of an EvidenceRecord can be described as follows:

1. Select a data object or group of data objects to archive.

2. Create the initial Archive Timestamp (see Section 4, "Archive
    Timestamp").

3. Refresh the Archive Timestamp when necessary, by Timestamp
    Renewal or Hash-Tree Renewal (see Section 5).

The process of generation depends on whether the Archive Timestamps
are generated, stored, and managed by a centralized instance. In the
case of central management, it is possible to collect many data
objects, build hash trees, store them, and reduce them later. In
case of local generation, it might be easier to generate a simple
Archive Timestamp without building hash trees. This can be

accomplished by omitting the reducedHashtree field from the
ArchiveTimestamp. In this case, the ArchiveTimestamp covers a single
data object. Using this approach, it is possible to "convert"
existing timestamps into ArchiveTimestamps for renewal.

3.3. Verification

The Verification of an EvidenceRecord overall can be described as
follows:

1. Select an archived data object or group of data objects

2. Re-encrypt data object/data object group, if encryption field is
    used (for details, see Section 6).

3. Verify Archive Timestamp Sequence (details in Section 4 and
    Section 5).

4. Archive Timestamp

An Archive Timestamp is a timestamp and a set of lists of hash
values. The lists of hash values are generated by reduction of an
ordered Merkle hash tree [MER1980]. The leaves of this hash tree are
the hash values of the data objects to be timestamped. Every inner
node of the tree contains one hash value, which is generated by
hashing the concatenation of the children nodes. The root hash
value, which represents unambiguously all data objects, is
timestamped.

4.1. Syntax

An Archive Timestamp has the following ASN.1 Syntax:

ASN.1 Archive Timestamp

```
ArchiveTimeStamp ::= SEQUENCE {
  digestAlgorithm [0] AlgorithmIdentifier OPTIONAL,
  attributes      [1] Attributes OPTIONAL,
  reducedHashtree [2] SEQUENCE OF PartialHashtree OPTIONAL,
  timeStamp       ContentInfo}

PartialHashtree ::= SEQUENCE OF OCTET STRING

Attributes ::= SET SIZE (1..MAX) OF Attribute
```

The fields of type ArchiveTimeStamp have the following meanings:

   digestAlgorithm identifies the digest algorithm and any associated
   parameters used within the reduced hash tree. If the optional field
   digestAlgorithm is not present, the digest algorithm of the timestamp
   MUST be used. Which means, if timestamps according to [RFC3161] are
   used in this case, the content of this field is identical to
   hashAlgorithm of messageImprint field of TSTInfo.

   attributes contains information an LTA might want to provide to
   document individual renewal steps and the creation of the individual
   ArchiveTimeStamps, e.g., applied policies. As the structure of the
   ArchiveTimeStamp may be protected by hash and timestamps, the
   ordering is relevant, which is why a SET is used instead of a
   SEQUENCE.

   reducedHashtree contains lists of hash values, organized in
   PartialHashtrees for easier understanding. They can be derived by
   reducing a hash tree to the nodes necessary to verify a single data
   object. Hash values are represented as octet strings. If the
   optional field reducedHashtree is not present, the ArchiveTimestamp
   simply contains an ordinary timestamp.

   timeStamp should contain the timestamp as defined in Section 1.3.
   (e.g., as defined with TimeStampToken in [RFC3161]). Other types of
   timestamp MAY be used, if they contain time data, timestamped data,
   and a cryptographically secure confirmation from the TSA of these
   data.

4.2. Generation

   The lists of hash values of an Archive Timestamp can be generated by
   building and reducing a Merkle hash tree [MER1980].

   Such a hash tree can be built as follows:

   1. Collect data objects to be timestamped.

   2. Choose a secure hash algorithm H and generate hash values for the
       data objects. These values will be the leaves of the hash tree.

   3. For each data group containing more than one document, its
       respective document hashes are binary sorted in ascending order,
       concatenated, and hashed. The hash values are the complete
       output from the hash algorithm, i.e., leading zeros are not
       removed, with the most significant bit first.

   4. If there is more than one hash value, place them in groups and
       sort each group in binary ascending order. Concatenate these
       values and generate new hash values, which are inner nodes of

     this tree. (If additional hash values are needed, e.g., so that
     all nodes have the same number of children, any data may be
     hashed using H and used.) Repeat this step until there is only
     one hash value, which is the root node of the hash tree.

5.   Obtain a timestamp for this root hash value. The hash algorithm
     in the timestamp request MUST be the same as the hash algorithm
     of the hash tree, or the digestAlgorithm field of the
     ArchiveTimeStamp MUST be present and specify the hash algorithm
     of the hash tree.

An example of a constructed hash tree for 3 data groups, where data
groups 1 and 3 only contain one document, and data group 2 contains 3
documents:

```
                  +------+
                  | h123 |
                  +------+
                 /        \
                /          \
           +----+          +----+
           | h12|          | h3 |
           +----+          +----+
           /    \
          /      \
     +----+  +-------+
     | h1 |  | h2abc |
     +----+  +-------+
               /  |  \
              /   |   \
             /    |    \
            /     |     \
       +----+  +----+  +----+
       | h2a|  | h2b|  | h2c|
       +----+  +----+  +----+
```
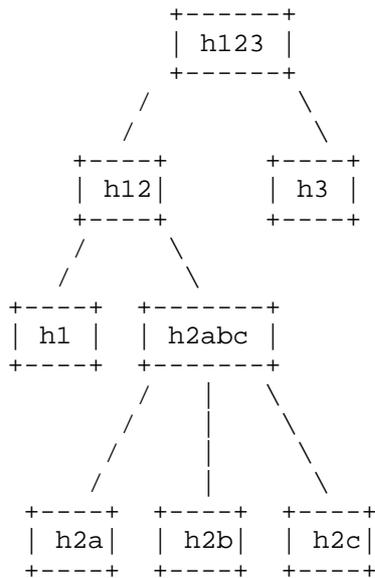
Figure 1: Hash tree

     h1 = H(d1) where d1 is the only data object in data group 1
     h3 = H(d3) where d3 is the only data object in data group 3
     h12 = H(binary sorted and concatenated (h1, h2abc))
     h123 = H(binary sorted and concatenated (h12, h3))
     h2a = H(first data object of data object group 2)
     h2b = H(second data object of data object group 2)
     h2c = H(third data object of data object group 2)
     h2abc = H(binary sorted and concatenated (h2a, h2b, h2c))

   The hash tree can be reduced to lists of hash values, necessary to
   have a proof of existence for a single data object:

   1. Generate hash value h of the data object, using hash algorithm H
       of the hash tree.

   2. Select all hash values, which have the same father node as h.
       Generate the first list of hash values by arranging these hashes,
       in binary ascending order. This will be stored in the structure
       of the PartialHashtree. Repeat this step for the father node of
       all hashes until the root hash is reached. The father nodes
       themselves are not saved in the hash lists -- they are
       computable.

   3. The list of all partialHashtrees finally is the reducedHashtree.
       (All of the specified hash values under the same father node,
       except the father node of the nodes below, are grouped in a
       PartialHashtree. The sequence list of all Partialhashtrees is
       the reducedHashtree.)

   4. Finally, add the timestamp and the info about the hash algorithm
       to get an Archive Timestamp.

   Assuming that the sorted binary ordering of the hashes in Figure 1
   is: h2abc < h1, then the reduced hash tree for data group 1 (d1) is:

```
       +--------------------------------+
       | +----------------+ +--------+ |
       | | +------+ +----+ | | +----+ | |
       | | | h2abc| | h1 | | | | h3 | | |
       | | +------+ +----+ | | +----+ | |
       | +----------------+ +--------+ |
       +--------------------------------+
```
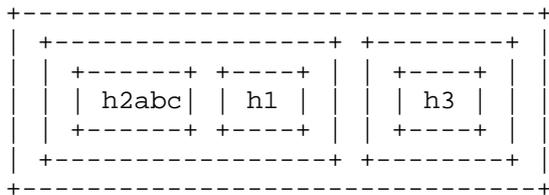
   Figure 2: Reduced hash tree for data group 1

      The pseudo ASN1 for this reduced hash tree rht1 would look like:
        rht1 = SEQ(pht1, pht2)

      with the PartialHashtrees pht1 and pht2
        pht1 = SEQ (h2abc, h1)
        pht2 = SEQ (h3)

Assuming the same hash tree as in Figure 1, the reduced hash tree for
all data objects in data group 2 is identical.

```
+-------------------------------------------------+
| +----------------------+  +--------+ +--------+ |
| | +----+ +----+ +----+ |  | +----+ | | +----+ | |
| | | h2b| | h2c| | h2a| |  | | h1 | | | | h3 | | |
| | +----+ +----+ +----+ |  | +----+ | | +----+ | |
| +----------------------+  +--------+ +--------+ |
+-------------------------------------------------+
```
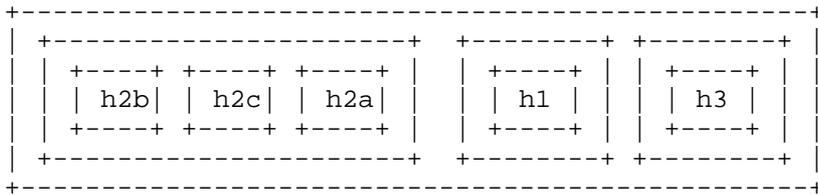
Figure 3: Reduced hash tree for data object group 2

   The pseudo ASN1 for this reduced hash tree would look like:
     rht2 = SEQ(pht3, pht4, pht5)

   with the PartialHashtrees pht3, pht4, and pht5
    pht3 = SEQ (h2b, h2c, h2a)
    pht4 = SEQ (h1)
    pht5 = SEQ (h3)

Note there are no restrictions on the quantity or length of hash-
value lists. Also note that it is profitable but not required to
build hash trees and reduce them. An Archive Timestamp may consist
only of one list of hash-values and a timestamp or only a timestamp
with no hash value lists.

The data (e.g. certificates, Certificate Revocation Lists (CRLs), or
Online Certificate Status Protocol (OCSP) responses) needed to verify
the timestamp MUST be preserved, and SHOULD be stored in the
timestamp itself unless this causes unnecessary duplication. A
timestamp according to [RFC3161] is a CMS object in which
certificates can be stored in the certificates field and CRLs can be
stored in the crls field of signed data. OCSP responses can be
stored as unsigned attributes [RFC3126]. Note [ANSI.X9-95.2005],
[ISO-18014-2.2002], and [ISO-18014-3.2004], which specify verifiable
timestamps that do not depend on certificates, CRLs, or OCSP
responses.

4.3. Verification

An Archive Timestamp shall prove that a data object existed at a
certain time, given by timestamp. This can be verified as follows:

1. Calculate hash value h of the data object with hash algorithm H
    given in field digestAlgorithm of the Archive Timestamp.

   2. Search for hash value h in the first list (partialHashtree) of
       reducedHashtree. If not present, terminate verification process
       with negative result.

   3. Concatenate the hash values of the actual list (partialHashtree)
       of hash values in binary ascending order and calculate the hash
       value h' with algorithm H. This hash value h' MUST become a
       member of the next higher list of hash values (from the next
       partialHashtree). Continue step 3 until a root hash value is
       calculated.

   4. Check timestamp. In case of a timestamp according to [RFC3161],
       the root hash value must correspond to hashedMessage, and
       digestAlgorithm must correspond to hashAlgorithm field, both in
       messageImprint field of timeStampToken. In case of other
       timestamp formats, the hash value and digestAlgorithm must also
       correspond to their equivalent fields if they exist.

   If a proof is necessary for more than one data object, steps 1 and 2
   have to be done for all data objects to be proved. If an additional
   proof is necessary that the Archive Timestamp relates to a data
   object group (e.g., a document and all its signatures), it can be
   verified additionally, that only the hash values of the given data
   objects are in the first hash-value list.

5. Archive Timestamp Chain and Archive Timestamp Sequence

   An Archive Timestamp proves the existence of single data objects or
   data object group at a certain time. However, this first Archive
   Timestamp in the first ArchiveTimeStampChain can become invalid, if
   hash algorithms or public key algorithms used in its hash tree or
   timestamp become weak or if the validity period of the timestamp
   authority certificate expires or is revoked.

   Prior to such an event, the existence of the Archive Timestamp or
   archive timestamped data has to be reassured. This can be done by
   creating a new Archive Timestamp. Depending on whether the timestamp
   becomes invalid or the hash algorithm of the hash tree becomes weak,
   two kinds of Archive Timestamp renewal are possible:

   o Timestamp Renewal: A new Archive Timestamp is generated, which
      covers the timestamp of the old one. One or more Archive
      Timestamps generated by Timestamp Renewal yield an Archive
      Timestamp Chain for a data object or data object group.

o Hash-Tree Renewal: A new Archive Timestamp is generated, which
   covers all the old Archive Timestamps as well as the data objects.
   A new Archive Timestamp Chain is started. One or more Archive
   Timestamp Chains for a data object or data object group yield an
   Archive Timestamp Sequence.

After the renewal, always only the last (i.e., most recent)
ArchiveTimeStamp and the algorithms and timestamps used by it must be
watched regarding expiration and loss of security.

5.1. Syntax

ArchiveTimeStampChain and ArchiveTimeStampSequence have the following
ASN.1 Syntax:

ASN.1 ArchiveTimeStampChain and ArchiveTimeStampSequence

ArchiveTimeStampChain   ::= SEQUENCE OF ArchiveTimeStamp

ArchiveTimeStampSequence ::= SEQUENCE OF
                             ArchiveTimeStampChain

ArchiveTimeStampChain and ArchiveTimeStampSequence MUST be ordered
ascending by time of timestamp. Within an ArchiveTimeStampChain, all
reducedHashtrees of the contained ArchiveTimeStamps MUST use the same
Hash-Algorithm.

5.2. Generation

The initial Archive Timestamp relates to a data object or a data
object group. Before cryptographic algorithms that are used within
the most recent Archive Timestamp (which is, at the beginning, the
initial one) become weak or their timestamp certificates become
invalid, Archive Timestamps have to be renewed by generating a new
Archive Timestamp.

In the case of Timestamp Renewal, the content of the timeStamp field
of the old Archive Timestamp has to be hashed and timestamped by a
new Archive Timestamp. The new Archive Timestamp MAY not contain a
reducedHashtree field, if the timestamp only simply covers the
previous timestamp. However, generally one can collect a number of
old Archive Timestamps and build the new hash tree with the hash
values of the content of their timeStamp fields.

The new Archive Timestamp MUST be added to the ArchiveTimestampChain.
This hash tree of the new Archive Timestamp MUST use the same hash
algorithm as the old one, which is specified in the digestAlgorithm

field of the Archive Timestamp or, if this value is not set (as it is
optional), within the timestamp itself.

In the case of Hash-Tree Renewal, the Archive Timestamp and the
archived data objects covered by the Archive Timestamp must be hashed
and timestamped again, as described below:

1. Select a secure hash algorithm H.

2. Select data objects d(i) referred to by initial Archive Timestamp
    (objects that are still present and not deleted). Generate hash
    values h(i) = H((d(i)). If data groups with more than one
    document are present, then one will have more than one hash for a
    group, i.e., h(i_a), h(i_b).., h(i_n)

3. atsc(i) is the encoded ArchiveTimeStampSequence, the
    concatenation of all previous Archive Timestamp Chains (in
    chronological order) related to data object d(i). Generate hash
    value ha(i) = H(atsc(i)).
    Note: The ArchiveTimeStampChains used are DER encoded, i.e., they
    contain sequence and length tags.

4. Concatenate each h(i) with ha(i) and generate hash values
    h(i)' = H (h(i)+ ha(i)). For multi-document groups, this is:
    h(i_a)' = H (h(i_a)+ ha(i))
    h(i_b)' = H (h(i_b)+ ha(i)), etc.

5. Build a new Archive Time Stamp for each h(i)'. (Hash-tree
    generation and reduction is defined in Section 4.2; note that
    each h(i)' will be treated in Section 4.2 as the document hash.
    The first hash value list in the reduced hash tree should only
    contain h(i)'. For a multi-document group, the first hash value
    list will contain the new hashes for all the documents in this
    group, i.e., h(i_a)', h(i_b)'.., h(i_n)')

6. Create new ArchiveTimeStampChain containing the new Archive
    Timestamp and append this ArchiveTimeStampChain to the
    ArchiveTimeStampSequence.

```
                     +-------+
                     | h123' |
                     +-------+
                     /       \
                    /         \
              +-----+         +----+
              | h12'|         | h3'|
              +-----+         +----+
              /     \
             /       \
         +----+  +--------+
         | h1'|  | h2abc' |
         +----+  +--------+
                 /   |   \
                /    |    \
               /     |     \
              /      |      \
          +----+  +----+  +----+
          |h2a'|  |h2b'|  |h2c'|
          +----+  +----+  +----+
```
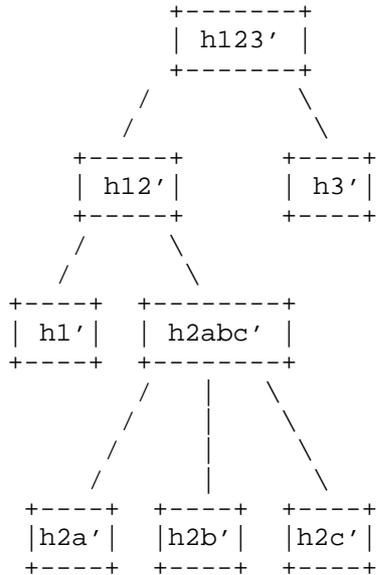
Figure 4: Hash tree from Hash-Tree Renewal

    Let H be the new secure hash algorithm
    ha(1), ha(2), ha(3) are as defined in step 4 above
    h1' = H(binary sorted and concatenated (H(d1), ha(1)))
      d1 is the original document from data group 1
    h3' = H(binary sorted and concatenated (H(d3), ha(3)))
      d3 is the original document from data group 3

    h2a = H(first data object of data object group 2)
     ...
    h2c = H(third data object of data object group 2)
    h2a' = H(binary sorted and concatenated (h2a, ha(2)))
     ...
    h2c' = H(binary sorted and concatenated (h2c, ha(2)))

    h2abc = H(binary sorted and concatenated (h2a', h2b', h2c'))

   ArchiveTimeStamps that are not necessary for verification should not
   be added to an ArchiveTimeStampChain or ArchiveTimeStampSequence.

5.3. Verification

   To get a non-repudiation proof that a data object existed at a
   certain time, the Archive Timestamp Chains and their relations to
   each other and to the data objects have to be proved:

1. Verify that the first Archive Timestamp of the first
    ArchiveTimestampChain (the initial Archive Timestamp) contains
    the hash value of the data object.

2. Verify each ArchiveTimestampChain. The first hash value list of
    each ArchiveTimeStamp MUST contain the hash value of the
    timestamp of the Archive Timestamp before. Each Archive
    Timestamp MUST be valid relative to the time of the following
    Archive Timestamp. All Archive Timestamps within a chain MUST
    use the same hash algorithm and this algorithm MUST be secure at
    the time of the first Archive Timestamp of the following
    ArchiveTimeStampChain.

3. Verify that the first hash value list (partialHashtree) of the
    first Archive Timestamp of all other ArchiveTimeStampChains
    contains a hash value of the concatenation of the data object
    hash and the hash value of all older ArchiveTimeStampChain.
    Verify that this Archive Timestamp was generated before the last
    Archive Timestamp of the ArchiveTimeStampChain became invalid.

In order to complete the non-repudiation proof for the data objects,
the last Archive Timestamp has to be valid at the time the
verification is performed.

If the proof is necessary for more than one data object, steps 1 and
3 have to be done for all these data objects. To prove the Archive
Timestamp Sequence relates to a data object group, verify that each
first Archive Timestamp of the first ArchiveTimeStampChain of the
ArchiveTimeStampSequence of each data object does not contain other
hash values in its first hash value list (than the hash values of the
other data objects).

6. Encryption

   If service providers are used to archive data and generate Archive
   Timestamps, it might be desirable or required that clients only send
   encrypted data to be archived. However, this means that evidence
   records refer to encrypted data objects. ERS directly protects the
   integrity of the bit-stream and this freezes the bit structure at the
   time of archiving. This precludes changing of the encryption scheme
   during the archival period, e.g., if the encryption scheme is no
   longer secure, a change is not possible without losing the integrity
   proof of the EvidenceRecord. In such cases, the services of a data
   transformation (and by this also possible re-encryption) done by a
   notary service might be a possible solution. To avoid problems when
   using the evidence records in the future, additional special
   precautions have to be taken:

   o Evidence generated to prove the existence of encrypted data cannot
      always be relied upon to prove the existence of unencrypted data.
      It may be possible to choose an algorithm or a key for decryption
      that is not the algorithm or key used for encryption. In this
      case, the evidence record would not be a non-repudiation proof for
      the unencrypted data. Therefore, only encryption methods should
      be used that make it possible to prove that archive-timestamped
      encrypted data objects unambiguously represent unencrypted data
      objects. All data necessary to prove unambiguous representation
      should be included in the archived data objects. (Note: In
      addition, the long-term security of the encryption schemes should
      be analyzed to determine if it could be used to create collision
      attacks.)

   o When a relying party uses an evidence record to prove the
      existence of encrypted data objects, it may be desirable for
      clients to only store the unencrypted data objects and to delete
      the encrypted copy. In order to use the evidence record, it must
      then be possible to unambiguously re-encrypt the unencrypted data
      to get exactly the data that was originally archived. Therefore,
      additional data necessary to re-encrypt data objects should be
      inserted into the evidence record by the client, i.e., the LTA
      never sees these values.

   An extensible structure is defined to store the necessary parameters
   of the encryption methods. The use of the specified
   encryptionInfoType and encryptionInfoValue may be heavily dependent
   on the mechanisms and has to be defined in other specifications.

6.1. Syntax

   The EncryptionInfo field in EvidenceRecord has the following syntax
   depending on the version of ASN.1.

6.1.1. EncryptionInfo in 1988 ASN.1

   1988 ASN.1 EncryptionInfo

   EncryptionInfo   ::=   SEQUENCE {
       encryptionInfoType   OBJECT IDENTIFIER,
       encryptionInfoValue   ANY DEFINED BY encryptionInfoType
   }

6.1.2. EncryptionInfo in 1997-ASN.1

   1997-ASN.1 EncryptionInfo

```
   EncryptionInfo         ::=        SEQUENCE {
      encryptionInfoType    ENCINFO-TYPE.&id
                                    ({SupportedEncryptionAlgorithms}),
      encryptionInfoValue   ENCINFO-TYPE.&Type
                  ({SupportedEncryptionAlgorithms}{@encryptionInfoType})
   }

   ENCINFO-TYPE ::= TYPE-IDENTIFIER

   SupportedEncryptionAlgorithms ENCINFO-TYPE ::= {...}
```

   encryptionInfo contains information necessary for the unambiguous
   re-encryption of unencrypted content so that it exactly matches with
   the encrypted data objects protected by the EvidenceRecord.

7. Security Considerations

   Secure Algorithms

   Cryptographic algorithms and parameters that are used within Archive
   Timestamps must 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-TS102176-1-2005]) have
   to be considered by verifying components. A generic solution for
   automatic interpretation of security suitability policies in
   electronic form is desirable but not the subject of this
   specification.

   Redundancy

   Retrospectively, certain parts of an Archive Timestamp may turn out
   to have lost their security suitability before this has been publicly
   known. For example, retrospectively, it may turn out that algorithms
   have lost their security suitability, and that even TSAs are
   untrustworthy. This can result in Archive Timestamps using those
   losing their probative force. Many TSAs are using the same signature
   algorithms. While the compromise of a private key will only affect
   the security of one specific TSA, the retrospective loss of security
   of a signature algorithm will have impact on a potentially large
   number of TSAs at once. To counter such risks, it is recommended to

generate and manage at least two redundant Evidence Records with
ArchiveTimeStampSequences using different hash algorithms and
different TSAs using different signature algorithms.

To best achieve and manage this redundancy, it is recommended to
manage the Archive Timestamps in a central LTA.

Secure Timestamps

Archive Timestamping depends upon the security of normal time
stamping. Security requirements for Time Stamping Authorities stated
in security policies have to be met. Renewed Archive Timestamps
should have the same or higher quality as the initial Archive
Timestamp. Archive Timestamps used for signature renewal of signed
data, should have the same or higher quality than the maximum quality
of the signatures.

Secure Encryption

For non-repudiation proof, it does not matter whether encryption has
been broken or not. Nevertheless, users should keep secret their
private keys and randoms used for encryption and disclose them only
if needed, e.g., in a lawsuit to a judge or expert. They should use
encryption algorithms and parameters that are prospected to be
unbreakable as long as confidentiality of the archived data is
important.

8. References

8.1. Normative References

   [CCITT.X208.1988]
             International Telephone and Telegraph Consultative
             Committee, "Specification of Abstract Syntax Notation One
             (ASN.1)", CCITT Recommendation X.208, November 1988.

   [CCITT.X209.1988]
             International Telephone and Telegraph Consultative
             Committee, "Specification of Basic Encoding Rules for
             Abstract Syntax Notation One (ASN.1)",
             CCITT Recommendation X.209, 1988.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
             Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato,
             "Internet X.509 Public Key Infrastructure Time-Stamp
             Protocol (TSP)", RFC 3161, August 2001.

   [RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet
              X.509 Public Key Infrastructure Certificate and
              Certificate Revocation List (CRL) Profile", RFC 3280,
              April 2002.

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

8.2. Informative References

   [ANSI.X9-95.2005]
              American National Standard for Financial Services,
              "Trusted Timestamp Management and Security", ANSI X9.95,
              June 2005.

   [CCITT.X680.2002]
              International Telephone and Telegraph Consultative
              Committee, "Abstract Syntax Notation One (ASN.1):
              Specification of basic notation", CCITT Recommendation
              X.680, July 2002.

   [CCITT.X690.2002]
              International Telephone and Telegraph Consultative
              Committee, "ASN.1 encoding rules: Specification of basic
              encoding Rules (BER), Canonical encoding rules (CER) and
              Distinguished encoding rules (DER)", CCITT Recommendation
              X.690, July 2002.

   [ETSI-TS102176-1-2005]
              European Telecommunication Standards Institute (ETSI),
              Electronic Signatures and Infrastructures (ESI);,
              "Algorithms and Parameters for Secure Electronic
              Signatures; Part 1: Hash functions and asymmetric
              algorithms", ETSI TS 102 176-1 V1.2.1, July 2005.

   [ISO-18014-1.2002]
              ISO/IEC JTC 1/SC 27, "Time stamping services - Part 1:
              Framework", ISO ISO-18014-1, February 2002.

   [ISO-18014-2.2002]
              ISO/IEC JTC 1/SC 27, "Time stamping services - Part 2:
              Mechanisms producing independent tokens", ISO ISO-18014-2,
              December 2002.

   [ISO-18014-3.2004]
              ISO/IEC JTC 1/SC 27, "Time stamping services - Part 3:
              Mechanisms producing linked tokens", ISO ISO-18014-3,
              February 2004.

   [MER1980]   Merkle, R., "Protocols for Public Key Cryptosystems,
               Proceedings of the 1980 IEEE Symposium on Security and
               Privacy (Oakland, CA, USA)", pages 122-134, April 1980.

   [NIST.800-57-Part1.2006]
               National Institute of Standards and Technology,
               "Recommendation for Key Management - Part 1: General
               (Revised)", NIST 800-57 Part1, May 2006.

   [RFC3126]   Pinkas, D., Ross, J., and N. Pope, "Electronic Signature
               Formats for long term electronic signatures", RFC 3126,
               September 2001.

   [RFC4810]   Wallace, C., Pordesch, U., and R. Brandner, "Long-Term
               Archive Service Requirements", RFC 4810, March 2007.

Appendix A. Evidence Record Using CMS

   An Evidence Record can be added to signed data or enveloped data in
   order to transfer them in a conclusive way. For CMS, a sensible
   place to store such an Evidence Record is an unsigned attribute
   (signed message) or an unprotected attribute (enveloped message).

   One advantage of storing the Evidence Record within the CMS structure
   is that all data can be transferred in one conclusive file and is
   directly connected. The documents, the signatures, and their
   Evidence Records can be bundled and managed together. The
   description in the appendix contains the normative specification of
   how to integrate ERS in CMS structures.

   The Evidence Record also contains information about the selection
   method that was used for the generation of the data objects to be
   timestamped. In the case of CMS, two selection methods can be
   distinguished:

   1. The CMS Object as a whole including contentInfo is selected as
       data object and archive timestamped. This means that a hash
       value of the CMS object MUST be located in the first list of hash
       values of Archive Timestamps.

   2. The CMS Object and the signed or encrypted content are included
       in the Archive Timestamp as separated objects. In this case, the
       hash value of the CMS Object as well as the hash value of the
       content have to be stored in the first list of hash values as a
       group of data objects.

   However, other selection methods could also be applied, for instance,
   as in [RFC3126].

   In the case of the two selection methods defined above, the Evidence
   Record has to be added to the first signature of the CMS Object of
   signed data. Depending on the selection method, the following Object
   Identifiers are defined for the Evidence Record:

   ASN.1 Internal EvidenceRecord Attribute

   id-aa-er-internal OBJECT IDENTIFIER ::= { iso(1) member-body(2)
      us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) id-aa(2) 49 }

   ASN.1 External EvidenceRecord Attribute

   id-aa-er-external OBJECT IDENTIFIER ::= { iso(1) member-body(2)
      us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) id-aa(2) 50 }

The attributes SHOULD only occur once. If they appear several times, they have to be stored within the first signature in chronological order.

If the CMS object doesn't have the EvidenceRecord Attributes -- which indicates that the EvidenceRecord has been provided externally -- the archive timestamped data object has to be generated over the complete CMS object within the existing coding.

In case of verification, if only one EvidenceRecord is contained in the CMS object, the hash value must be generated over the CMS object without the one EvidenceRecord. This means that the attribute has to be removed before verification. The length of fields containing tags has to be adapted. Apart from that, the existing coding must not be modified.

If several Archive Timestamps occur, the data object has to be generated as follows:

o During verification of the first (in chronological order) EvidenceRecord, all EvidenceRecord have to be removed in order to generate the data object.

o During verification of the nth one EvidenceRecord, the first n-1 attributes should remain within the CMS object.

o The verification of the nth one EvidenceRecord must result in a point of time when the document must have existed with the first n attributes. The verification of the n+1th attribute must prove that this requirement has been met.

Appendix B. ASN.1-Module with 1988 Syntax

    ASN.1-Module

    ERS {iso(1) identified-organization(3) dod(6)
         internet(1) security(5) mechanisms(5)
         ltans(11) id-mod(0) id-mod-ers88(2) id-mod-ers88-v1(1) }
    DEFINITIONS IMPLICIT TAGS ::=
    BEGIN

    -- EXPORTS ALL --

    IMPORTS

      -- Imports from RFC 3852 Cryptographic Message Syntax
    ContentInfo, Attribute

```
      FROM CryptographicMessageSyntax2004 -- FROM [RFC3852]
       { iso(1) member-body(2) us(840) rsadsi(113549)
         pkcs(1) pkcs-9(9) smime(16) modules(0) cms-2004(24) }

   -- Imports from RFC 3280 [RFC3280], Appendix A.1
 AlgorithmIdentifier
     FROM PKIX1Explicit88
         { iso(1) identified-organization(3) dod(6)
           internet(1) security(5) mechanisms(5) pkix(7)
           mod(0) pkix1-explicit(18) }
 ;

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

   EvidenceRecord ::= SEQUENCE {
      version                   INTEGER { v1(1) } ,
      digestAlgorithms          SEQUENCE OF AlgorithmIdentifier,
      cryptoInfos               [0] CryptoInfos OPTIONAL,
      encryptionInfo            [1] EncryptionInfo OPTIONAL,
      archiveTimeStampSequence  ArchiveTimeStampSequence
      }

   CryptoInfos ::= SEQUENCE SIZE (1..MAX) OF Attribute

   ArchiveTimeStamp ::= SEQUENCE {
     digestAlgorithm [0] AlgorithmIdentifier OPTIONAL,
     attributes      [1] Attributes OPTIONAL,
     reducedHashtree [2] SEQUENCE OF PartialHashtree OPTIONAL,
     timeStamp       ContentInfo}

   PartialHashtree ::= SEQUENCE OF OCTET STRING

   Attributes ::= SET SIZE (1..MAX) OF Attribute

   ArchiveTimeStampChain    ::= SEQUENCE OF ArchiveTimeStamp

   ArchiveTimeStampSequence ::= SEQUENCE OF
                                ArchiveTimeStampChain

   EncryptionInfo        ::=     SEQUENCE {
```

```
        encryptionInfoType      OBJECT IDENTIFIER,
        encryptionInfoValue     ANY DEFINED BY encryptionInfoType}

    id-aa-er-internal  OBJECT IDENTIFIER ::= { iso(1) member-body(2)
       us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) id-aa(2) 49 }

    id-aa-er-external  OBJECT IDENTIFIER ::= { iso(1) member-body(2)
       us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) id-aa(2) 50 }

    END
```

Appendix C. ASN.1-Module with 1997 Syntax

```
    ASN.1-Module

    ERS {iso(1) identified-organization(3) dod(6)
         internet(1) security(5) mechanisms(5)
         ltans(11) id-mod(0) id-mod-ers(1) id-mod-ers-v1(1) }
    DEFINITIONS IMPLICIT TAGS ::=
    BEGIN

    -- EXPORTS ALL --

    IMPORTS

     -- Imports from PKCS-7
    ContentInfo
        FROM PKCS7
            {iso(1) member-body(2) us(840) rsadsi(113549)
            pkcs(1) pkcs-7(7) modules(0)}

      -- Imports from AuthenticationFramework
    AlgorithmIdentifier
        FROM AuthenticationFramework
            {joint-iso-itu-t ds(5) module(1)
            authenticationFramework(7) 4}


     -- Imports from InformationFramework
    Attribute
        FROM InformationFramework
            {joint-iso-itu-t ds(5) module(1)
            informationFramework(1) 4}
    ;

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

```
   EvidenceRecord ::= SEQUENCE {
      version                   INTEGER { v1(1) } ,
      digestAlgorithms          SEQUENCE OF AlgorithmIdentifier,
      cryptoInfos               [0] CryptoInfos OPTIONAL,
      encryptionInfo            [1] EncryptionInfo OPTIONAL,
      archiveTimeStampSequence  ArchiveTimeStampSequence
      }

   CryptoInfos ::= SEQUENCE SIZE (1..MAX) OF Attribute
           (WITH COMPONENTS {
               ...,
               valuesWithContext   ABSENT
            })

   ArchiveTimeStamp ::= SEQUENCE {
     digestAlgorithm [0] AlgorithmIdentifier OPTIONAL,
     attributes      [1] Attributes OPTIONAL,
     reducedHashtree [2] SEQUENCE OF PartialHashtree OPTIONAL,
     timeStamp       ContentInfo}

   PartialHashtree ::= SEQUENCE OF OCTET STRING

   Attributes ::= SET SIZE (1..MAX) OF Attribute
           (WITH COMPONENTS {
               ...,
               valuesWithContext   ABSENT
            })

   ArchiveTimeStampChain    ::= SEQUENCE OF ArchiveTimeStamp

   ArchiveTimeStampSequence ::= SEQUENCE OF
                                ArchiveTimeStampChain

   EncryptionInfo        ::=       SEQUENCE {
      encryptionInfoType    ENCINFO-TYPE.&id
                                   ({SupportedEncryptionAlgorithms}),
      encryptionInfoValue   ENCINFO-TYPE.&Type
                ({SupportedEncryptionAlgorithms}{@encryptionInfoType})
   }

   ENCINFO-TYPE ::= TYPE-IDENTIFIER

   SupportedEncryptionAlgorithms ENCINFO-TYPE ::= {...}

   id-aa-er-internal  OBJECT IDENTIFIER ::= { iso(1) member-body(2)
      us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) id-aa(2) 49 }

   id-aa-er-external  OBJECT IDENTIFIER ::= { iso(1) member-body(2)
```

```
      us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) id-aa(2) 50 }
```

    END

Authors' Addresses

    Tobias Gondrom
    Open Text Corporation
    Werner-von-Siemens-Ring 20
    Grasbrunn, Munich   D-85630
    Germany

    Phone: +49 (0) 89 4629-1816
    Fax: +49 (0) 89 4629-33-1816
    EMail: tobias.gondrom@opentext.com

    Ralf Brandner
    InterComponentWare AG
    Industriestra?e 41
    Walldorf   D-69119
    Germany

    EMail: ralf.brandner@intercomponentware.com

    Ulrich Pordesch
    Fraunhofer Gesellschaft
    Rheinstra?e 75
    Darmstadt   D-64295
    Germany

    EMail: ulrich.pordesch@zv.fraunhofer.de

Full Copyright Statement

Intellectual Property

Acknowledgement