Internet Draft
draft-ietf-pkix-tap-00.txt
February 2003
Expires August 2003

C. Wallace
CygnaCom Solutions
S. Chokhani
Orion Security

Trusted Archive Protocol (TAP)

Status of this Memo

Copyright Notice

Abstract

A Trusted Archive Authority (TAA) is a service that supports long-term non-repudiation by maintaining secure storage of cryptographically refreshed information. This document defines a set of transactions for interacting with a Trusted Archive Authority (TAA) and establishes a means of representing archived information.

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

A Trusted Archive Authority (TAA) is a service that supports long-term non-repudiation by maintaining secure storage of cryptographically refreshed information. This document defines a trusted archive protocol (TAP) that provides a set of transactions for interactions with a TAA (i.e. submission, retrieval and deletion of information).

1.1 Terminology

A TAA generates and maintains various data as part of the archive process. Throughout this document, entities submitting data to the TAA for archival are referred to as submitters and entities requesting retrieval or deletion of data are referred to as requestors. This document uses the following terms to describe the artifacts of the archive process:

Archived data: archived data is the data presented to the TAA by the submitter.

Archive token: an archive token is an object generated by the TAA when data is submitted and accepted for archiving. The archive token is returned to the submitter and may be used to request retrieval or deletion of the archived data and associated cryptographic information. For purposes of future retrieval or deletion, applications may treat the archive token as an opaque blob. The archive token includes: submitter DN, timestamp token, TAA date and time upon submission and, optionally, tracking information. To verify the accuracy of information archived by the TAA, submitters MUST verify the contents of the archive token as described below in section 3.

Archive record: an archive record contains the cryptographic refresh history compiled by the TAA. The initial archive record is the timestamp token obtained for the submitted data. The timestamp token format is defined in [RFC3161] and consists of a ContentInfo object containing a TSTInfo object. Upon each refresh, the most recent archive record becomes the prevArchRecord field of a new TimeStampedData object, a timestamp is obtained for the TimeStampedData object and is placed in the timestamp field of a new ArchiveRecordData and the entire ArchiveRecordData structure placed in a ContentInfo object. The ContentInfo object serves as the new archive record. When verifying an archive record, verification terminates when the original timestamp token is verified against the archived data.

Archive package: an archive package is an object containing, minimally, the archive token, archive record and archived data. The archive package MAY include additional cryptographic information.

1.2 Data

A TAA may be able to archive any data format or a TAA MAY implement features that limit the types of data that will be accepted for archiving. A TAA MAY implement additional support for some data formats, e.g. a TAA could implement a CMS message verification feature. Additional features SHOULD be implemented using an archive control.

Data submitted to a TAA MAY include all, some or none of the cryptographic information necessary for long-term dispute resolution. Archived data MAY be submitted with or without type information and/or instructions that request the TAA to act upon the data prior to archival, i.e. an archive control. A TAA MAY implement features that assist in the collection and/or validation of cryptographic information or otherwise act upon the submitted data. Submitters MAY submit data that is thought to be cryptographically valid or invalid. Retrieval clients MAY submit information sufficient to identify 0 or more archive records for retrieval.

1.3 Entities

This specification identifies four entities involved in TAP: TAA, timestamp authority (TSA), submission client and retrieval/deletion client. The TAA MAY be aware of the archived data format or not aware. The submission client MAY be TAP-aware or non-TAP-aware. The retrieval/deletion client MUST be TAP-aware and MAY be aware of the archived data format or not aware. The TSA MAY be independent of the TAA (i.e. the TAA acts as a timestamp client) or the TAA MAY be a TSA.

The provision for non-TAP-aware submission clients is intended to support simple, existing clients, such as a FTP client. In such cases a TAP-aware client should be used to process the TAA response, which may be delivered via a different transport.

TAA certificates MUST include an instance of the extendedKeyUsage extension permitting operation as a TAA. The value must be set to id-kp-trustedArchive.

1.4 Services

A TAA MUST provide the following services:

- Archived data preservation

- Archive token generation (including acquisition of a timestamp for the archived data)
- Periodic refresh of archive record
- Trusted cryptographic information preservation for verification of an archive record (i.e. trust anchors, certificates, CRLs, OCSP responses, OCSP responder certificates, etc.)
- Archive package retrieval and deletion

A TAA MAY provide additional, optional services, for example:

- Historical trust anchor preservation
- PKI information collection and/or validation
- Cryptographic message validation

Like the RFC on Electronic Signature Formats for long-term electronic signatures [RFC3126], this draft relies on CMS [RFC3369] and the Time Stamp Protocol [RFC3161]. This specification defines the following:

- A data transfer protocol between TAAs and clients
- Artifacts that can be used to archive and preserve any cryptographic service, such as digital signatures, and to archive any non-cryptographic data.

TAP uses a timestamp refresh approach that greatly reduces (and can be used to eliminate) the need to trust the TAA for the integrity of the archive data. In other words, data modifications made to the archive records by the TAA can be detected.

1.5 Applications

The TAA can be unaware of the data being archived and can be used to archive cryptographic data or non-cryptographic data. Cryptographic data can be signed, encrypted or both.

In support of long-term preservation of digital signatures, submitters can package all the certificates, revocation information (CRLs and OCSP responses) and, optionally, trust anchors in the submitted data to facilitate signature verification at any time in the future without needing the services of a repository or other source for certificates and revocation information. If the retrieval client uses another trusted source for trust anchors for signature verification and for trusted timestamp verification, then the TAA need not be trusted for the integrity of the data. The TSA MUST be trusted in all cases.

2. Trusted Archive Protocol

The following sections describe the transaction formats that comprise the TAP. Submission and retrieval requests sent to a TAA MAY be signed, not authenticated or authenticated using other means such as

client-authenticated SSL/TLS. Deletion requests MUST be authenticated. Messages sent from a TAA are always signed using the CMS SignedData ([RFC3369]) format with a TAP response payload. All response messages from a TAA MUST be signed and MUST NOT contain any signatures other than the signature of the TAA.

Unsigned requests consist of an ArchiveSubmissionReq, ArchiveRetrievalReq or ArchiveDeletionReq encapsulated in a ContentInfo object. An overview of this structure is provided below. Many details are not shown, but the way that TAP makes use of CMS is clearly illustrated.

```
ContentInfo {
  contentType, -- id-tap-archiveReq, id-tap-archiveRetrievalReq
               -- or id-tap-archiveDeletionReq

  content      -- ArchiveSubmissionReq, ArchiveRetrievalReq
               -- or ArchiveDeletionReq
}
```

Signed requests and signed responses consist of an ArchiveSubmissionReq, ArchiveRetrievalReq, ArchiveDeletionReq, ArchiveSubOrDelResp or ArchiveRetrievalResp encapsulated in a SignedData, which is in turn encapsulated in a ContentInfo. An overview of this structure is provided below. Again, many details are not shown, but the way that TAP makes use of CMS is clearly illustrated.

```
ContentInfo {
  contentType, -- id-signedData (1.2.840.113549.1.7.2)
  content      -- SignedData
}

SignedData {
  version,
  digestAlgorithms,
  encapContentInfo,  -- contents as described below
  certificates,      -- (Optional)
  crls,              -- (Optional)
  signerInfos        -- (only one in TAP)
}

SignerInfo {
  version,
  sid,
  digestAlgorithm,
  signedAttrs,       -- (Required)
  signatureAlgorithm,
  signature,
```

unsignedAttrs

```
}

EncapsulatedContentInfo {
   eContentType,  -- id-tap-archiveReq,
                  -- id-tap-archiveRetrievalReq,
                  -- id-tap-archiveDeletionReq,
                  -- id-tap-archiveSubOrDelResp or
                  -- id-tap-archiveRetrievalResp

   eContent    -- OCTET STRING containing
               -- ArchiveSubmissionReq, ArchiveRetrievalReq,
               -- ArchiveDeletionReq, ArchiveSubOrDelResp or
               -- ArchiveRetrievalResp
}
```

The syntaxes for SignedData and ContentInfo are defined in [RFC3369]. The syntaxes for all request and response types are defined below.

For all response messages, the TAA server MUST include its own certificate in the certificates field within SignedData. Other certificates MAY be included. The TAA server MAY provide one or more CRLs in the crls field within SignedData. The signedAttrs within SignerInfo MUST include the content-type and message-digest attributes defined in [RFC3369] (because the content type of the EncapsulatedContentInfo value is not id-data).

2.1 Archive submission request format

Archive submission requests are defined as follows:

```
ArchiveSubmissionReq ::= SEQUENCE
{
   version           TAPVersion DEFAULT v1,
   submitterName     GeneralName,
   policy            OBJECT IDENTIFIER    OPTIONAL,
   archiveControls   [0] ArchiveControls  OPTIONAL,
   archivedData      ArchivedData
}
```

TAA implementations MAY require authentication via CMS, SSL/TLS, or other means. TAA implementations MAY support alternative submission formats in addition to ArchiveSubmissionReq.

2.1.1 version

The version field (currently v1) describes the version of the archive submission request.

```
TAPVersion ::= INTEGER { v1(0) }
```

2.1.2 submitterName

The submitterName field identifies the entity submitting the associated data for archiving. If authentication is performed, TAA implementations SHOULD confirm that the value in the submitterName field is consistent with authenticated information. For successful requests, TAAs MUST include the submitterName contained in a request in the resulting archive token.

2.1.3 policy

The policy field, if present, indicates the policy under which the archive service SHOULD operate with regard to the data submitted as part of the request.

2.1.4 archiveControls

The archiveControls field may be used to request the TAA to perform additional actions, for example, server-side validation of the data field of archiveData or inclusion of a nonce in the response.

TAAs MUST reject requests containing unrecognized or unsupported archive controls. Archive controls SHOULD be defined such that for each control included in a request a corresponding control is included in the response.

```
ArchiveControls ::= SEQUENCE SIZE (1..MAX) OF ArchiveControl
ArchiveControl ::= SEQUENCE
{
   archiveControlType   OBJECT IDENTIFIER
   archiveControlValue  ANY DEFINED BY archiveControlType OPTIONAL
}
```

2.1.5 archivedData

The archivedData field contains the data to be archived and, optionally, type information. The type field of archivedData is advisory and is for use when processing archiveControls and/or for use by retrieval/deletion clients. The data field of archivedData contains the data to archive.

```
ArchivedData ::= SEQUENCE
{
   type     ArchivedDataType  OPTIONAL,
```

```
   data     OCTET STRING
}

ArchivedDataType ::= CHOICE
{
   oid      OBJECT IDENTIFIER,
   mimeType UTF8String
}
```

When type information is included in a submission request, TAAs SHOULD return type information in future retrieval responses containing the associated archived data.

2.2 Archive submission response format

Archive submission responses are defined as follows:

```
ArchiveSubOrDelResp ::= SEQUENCE
{
   version           TAPVersion DEFAULT v1,
   status            ArchiveStatus,
   archiveToken      ArchiveToken         OPTIONAL,
   archiveControls   [0] ArchiveControls  OPTIONAL
}
```

ArchiveSubOrDelResp objects MUST be returned in the eContent field of a CMS SignedData message.

2.2.1 version

The version field (currently v1) describes the version of the archive submission response.

2.2.2 status

The status field indicates the outcome of request processing and is comprised of a status code and an optional status string.

```
ArchiveStatus ::= SEQUENCE
{
   code           ArchiveStatusCode,
   statusString   UTF8String  OPTIONAL
}

ArchiveStatusCode ::= ENUMERATED
{
   success              (0),  -- success
   genericFailure       (1),  -- misc. unspecified failure
```

```
   authenticationFailed (2),  -- authentication failed (or absent)
   unauthorizedRequest  (3),  -- submitter(or request) not authorized
   unrecognizedControl  (4),  -- unrecognized or disallowed control
   controlFailure       (5),  -- control processing failed
   policyFailure        (6),  -- policy not supported
   timestampFailure     (7),  -- timestamp could not be obtained
   retrievalDelayed     (8),-- retrieval may require manual action
   unsupportedDataFormat(9) -- format of submitted data not supported
   -- add more status codes
}
```

2.2.3 archiveToken

The archiveToken field contains information that can be used to request retrieval or deletion of the archived data in the future. An archiveToken MUST be included in all successful submission responses. Submitters MUST verify archive tokens as described in section 3 to ensure that the archive token accurately reflects the submitted data, i.e. the values in the submitterName, curTime and timestamp fields are consistent with request.

```
ArchiveToken ::= ContentInfo
 -- content type: id-tap-archiveToken
 -- content: ArchiveTokenData

ArchiveTokenData ::= SEQUENCE
{
   submitterName  GeneralName,
   timestamp      TimeStampToken,
   curTime        GeneralizedTime,
   trackingInfo   TrackingInfos  OPTIONAL
}

TrackingInfos ::= SEQUENCE SIZE (1..MAX) OF TrackingInfo
TrackingInfo ::= ContentInfo
```

The submitterName field contains the value from the submitterName field in the request.

The timestamp field contains a timestamp generated for the archived data.

The curTime field contains the TAA time when the archive token was created.

The trackingInfo field, if present, MAY contain information relevant only to the TAA and/or MAY contain information that identifies the TAA, i.e. a URL. Submission clients and retrieval/deletion clients

are not required to process the contents of the trackingInfo field but SHOULD be capable of processing the TAALocation TrackingInfo.

2.2.4 archiveControls

The archiveControls field is used to return information associated with a control included in the request, for example, the outcome of server-side validation or a nonce from the request. TAAs MUST NOT include controls in a response that are not associated with controls in a request. Submission clients SHOULD be able to process controls in accordance with the control definition.

2.3 Archive retrieval request format

Archive retrieval requests are defined as follows:

```
ArchiveRetrievalReq ::= SEQUENCE
{
   version           TAPVersion DEFAULT v1,
   requestorName     GeneralName,
   retrievalRequest  ArchiveRetrievalInfo OPTIONAL,
   archiveControls   [0] ArchiveControls  OPTIONAL
}
```

The request includes information identifying the archived data to retrieve or to initiate a search.

TAA implementations MAY require authentication via CMS, SSL/TLS, or other means.

2.3.1 version

The version field (currently v1) describes the version of the archive retrieval request.

2.3.2 requestorName

The requestorName field identifies the entity requesting retrieval of an archive package. If authentication is performed, TAA implementations SHOULD confirm that the value in the requestorName field is consistent with authenticated information.

2.3.3 retrievalRequest

The ArchiveRetrievalInfo structure permits clients to fully identify an archive using an archive token, to initiate a search using a partial set of information or to complete a delayed request using a poll reference. The retrievalRequest field may be omitted when the archiveControls field contains all necessary information, such as when requesting only trust anchor information via a TrustAnchorRequest control.

```
ArchiveRetrievalInfo ::= CHOICE
{
   archiveToken   [0] ArchiveToken,
   archiveInfo    [1] ArchiveInfo,
   pollReference  [2] OCTET STRING
}
```

The archiveToken field can be used to identify a specific archive package for retrieval. ArchiveRetrievalResps associated with ArchiveRetrievalReqs containing an archive token MUST contain a single ArchivePackage. The archiveInfo field can be used to retrieve a collection of tokens or archive packages. The pollReference field can be used to complete a delayed request. The value included in pollReference is the value returned by the TAA in an ArchiveRetrievalResp with status set to retrievalDelayed.

```
ArchiveInfo::= SEQUENCE
{
   tokensOnly     BOOLEAN              DEFAULT TRUE,
   submitterName  [0] GeneralName      OPTIONAL,
   timestamp      [1] TimeStampToken   OPTIONAL,
   timeInfo       [2] ArchiveTimeInfo  OPTIONAL,
}

ArchiveTimeInfo ::= SEQUENCE
{
   time     GeneralizedTime,
   accuracy Accuracy OPTIONAL
}
```

The tokensOnly field of ArchiveInfo can be used to avoid retrieving data and cryptographic information for each archive that matches the query. The fields of ArchiveInfo can be used to query for archive tokens or archive packages that match the specified search parameters.

The accuracy field in ArchiveTimeInfo is applied to the time field to define a range of time used when searching. Accuracy is defined in [RFC3161].

2.3.4 archiveControls

The archiveControls field may be used to request additional, optional services from a TAA, such as a limit on the number of returned results, a nonce or an indication to return trust anchors known to the TAA at the time an archive was created.

TAAs MUST reject requests containing unrecognized or unsupported archive controls.

2.4 Archive retrieval response

Archive retrieval responses are defined as follows:

```
ArchiveRetrievalResp ::= SEQUENCE
{
   version           TAPVersion DEFAULT v1,
   status            ArchiveStatus,
   archiveControls   [0] ArchiveControls     OPTIONAL,
   results           ArchiveRetrievalResults OPTIONAL
}
```

ArchiveRetrievalResp objects MUST be returned as the eContent field of a CMS SignedData message.

2.4.1 version

The version field (currently v1) describes the version of the archive retrieval response.

2.4.2 status

The status field indicates that outcome of the request processing and is comprised of a status code and an optional status string.

2.4.3 archiveControls

The archiveControls field is used to return information associated with a control included in the request. TAAs MUST NOT include controls in a response that are not associated with controls in a request. Retrieval/deletion clients SHOULD be able to process controls in accordance with the control definition.

2.4.4 results

The results of a successful retrieval request are returned as a sequence of at least one ArchivePackage, which contains the archive token and (optionally) the archive package data. A pollReference MAY be returned in cases where the archive package is not immediately available, for example, when manual intervention is required to retrieve an archive.

```
ArchiveRetrievalResults ::= SEQUENCE SIZE (1..MAX) OF ArchivePackage

ArchivePackage ::= SEQUENCE
{
   archiveToken      ArchiveToken,
   packageData       [0] ArchivePackageData  OPTIONAL,
   pollReference     [1] OCTET STRING        OPTIONAL
}

ArchivePackageData ::= SEQUENCE
{
   digestAlgorithms  DigestAlgorithmIdentifiers,
   policy            OBJECT IDENTIFIER          OPTIONAL,
   archiveRecord     ArchiveRecord,
   cryptoInfos       [0] CryptoInfos            OPTIONAL,
   archivedData      ArchivedData
}
```

The digestAlgorithms field identifies all digest algorithms that were applied to the archived data over the lifetime of the archive record. To successfully verify all archive record components, the archived data MUST be hashed using each of the algorithms identified in the digestAlgorithms field.

The archiveRecord field contains a nested structure with the complete refresh history for the archived data. TAAs SHOULD store all cryptographic information necessary to verify each layer of the archive record in the certificates, crls and unsignedAttrs fields of the timestamp token, i.e. each timestamp token in the history SHOULD be self-contained for validation purposes under protection of the next layer in the archive record. A CryptoInfos unsignedAttrs field MAY be used to convey OCSP responses and/or trust anchor information. The object identifier id-tap-cryptoInfos identifies the CryptoInfos attribute. CryptoInfos attribute values have the ASN.1 type CryptoInfos.

```
ArchiveRecord ::= ContentInfo
-- content type: id-tap-archiveRecordData
-- content: ArchiveRecordData
```

```
ArchiveRecordData ::= SEQUENCE
{
   timestampedData   TimeStampedData,  -- covered by timestamp
   timestamp         TimeStampToken
}

TimeStampedData ::= SEQUENCE
{
   prevArchRecord ContentInfo,   -- previous record
   messageImprint MessageImprint -- hash of archived data
}
```

The cryptoInfos field contains additional information that may be useful when verifying the archived data. This information may be included as a service by a TAA or due to collection of information requested via an archive control, etc. Retrieval/deletion clients are free to ignore any or all CryptoInfos contained in an archive package.

```
CryptoInfos ::= SEQUENCE SIZE (1..MAX) OF CryptoInfo
CryptoInfo ::= SEQUENCE
{
   cryptoInfoType    OBJECT IDENTIFIER
   cryptoInfoValue   ANY DEFINED BY cryptoInfoType
}
```

The archivedData field contains the data that was submitted to the TAA and, optionally, type information. The data field within the ArchivedData structure contains the data to hash using the algorithms identified in the digestAlgorithms field of ArchivePackageData.

2.5 Archive deletion request

Archive deletion requests are defined as follows:

```
ArchiveDeletionReq ::= SEQUENCE
{
   version           TAPVersion DEFAULT v1,
   requestorName     GeneralName,
   archiveToken      ArchiveToken,
   archiveControls   [0] ArchiveControls  OPTIONAL
}
```

The request includes information identifying the archived data to delete. Deletion requests MUST be authenticated.

2.5.1 version

The version field (currently v1) describes the version of the archive deletion request.

2.5.2 requestorName

The requestorName field identifies the entity requesting retrieval of an archive package. If authentication is performed, TAA implementations SHOULD confirm that the value in the requestorName field is consistent with authenticated information.

2.5.3 archiveToken

The archive token field identifies the archived data to delete.

2.5.4 archiveControls

The archiveControls field may be used to request additional, optional services from a TAA.

TAAs MUST reject requests containing unrecognized or unsupported archive controls.

2.6 Archive deletion response

Archive deletion responses are of type ArchiveSubOrDelResp as defined above. The meaning of each field in the context of a deletion response is described below.

ArchiveSubOrDelResp objects MUST be returned in the eContent field of a CMS SignedData message.

2.6.1 version

The version field (currently v1) describes the version of the archive deletion response.

2.6.2 status

The status field indicates that outcome of the request processing and is comprised of a status code and an optional status string.

2.6.3 archiveToken

The archiveToken field contains information identifying the deleted archive data. Successful responses MUST include an archiveToken identifying the archive that was deleted. The archiveToken MUST match the archiveToken contained in the deletion request.

2.6.4 archiveControls

The archiveControls field is used to return information associated with a control included in the request. TAAs MUST NOT include controls in a response that are not associated with controls in a request. Retrieval/deletion clients SHOULD be able to process controls in accordance with the control definition.

3. Validation

The signature on all TAA responses MUST be verified. TAA signatures on protocol transactions should be verified using current trust anchors known to the client. This section discusses additional validation steps for each type of transaction.

3.1 Submission

After verifying the signature of a successful ArchiveSubOrDelResp, compliant submission clients MUST perform the following processing of the submission response contents:

- Process each archive control per definition of control;
- Verify the signature of the Time Stamp Authority (TSA) on the timestamp token contained in the archive token;
- Verify that the hash contained in the timestamp token represents the hash of the data submitted by the client to the TAA; and
- Verify that the time on the timestamp token is reasonably close to the current time.

Verification that the curTime in the archive token is reasonably close to the current time is RECOMMENDED and confirmation that the submitterName is correct is RECOMMENDED.

For the signature verification of the TSA, the submitter can choose to use the trust anchors returned by the TAA, if present, or rely on its own list of trust anchors.

Controls that involve TAA-alteration of submitted data, i.e. collection and inclusion of relevant cryptographic information in the submitted data, may impact the verification of the timestamp field.

3.2 Retrieval

After verifying the signature of a successful ArchiveRetrievalResp, compliant retrieval/deletion clients MUST perform the following processing of the retrieval response contents:

- If an archive token was included in the request, the archive token the ArchivePackage should be compared with the requested archive token;
- Process each archive control per definition of control;
- Hash the data field of the archived data using each of the algorithms identified in the digestAlgorithms field of the ArchivePackage data structure;
- Verify the outermost timestamp token;
- Verify that the timestamp on the outermost token is current;
- Verify all remaining timestamp tokens; and
- Verify that in each instance a new timestamp token was applied prior to the preceding timestamp token expiry.

See the security considerations section for additional information regarding selection of the trust anchors to be used for timestamp token verification.

The verification of the archived data is beyond the scope of this specification. This specification provides mechanism to carry all the data required to make such verification possible, but the TAA need not be aware of the data format. For example, if a submitter submits a signed CMS message with all the certificates, revocation information (CRLs and OCSP responses), and trust anchors required to verify the message, that message could be verified upon retrieval to prove that the signature was valid at the time of the inner most timestamp on the retrieved data.

The archiveRecord MUST be verified as described above by verifying the timestamp present in each layer of the ArchiveRecord structure. Layers should be validated in turn beginning with the outermost layer and ending with the innermost layer. The innermost layer is simply the timestamp obtained for the archived data; outer layers are ArchiveRecord structures, which contain the previous record, a hash of the archived data and a timestamp. When verifying the innermost timestamp, verify that the hash contained in the timestamp token represents the hash of the archived data. When verifying outer timestamps, verify that the hash contained in the timestamp token matches the hash of the corresponding TimeStampedData structure and that the hash contained in the TimeStampedData structure represents a hash of the archived data.

Timestamp token signatures MAY be verified using client-obtained

trust anchor and revocation information or using information provided

by the TAA. TAAs MAY provide relevant cryptographic information in the CryptoInfos unsigned attribute of the SignerInfo structure and the certificates and crls fields of the SignedData structure of each timestamp token.

ArchiveRecord verification terminates when the innermost ContentInfo object containing a timestamp token (covering the archived data) is verified.

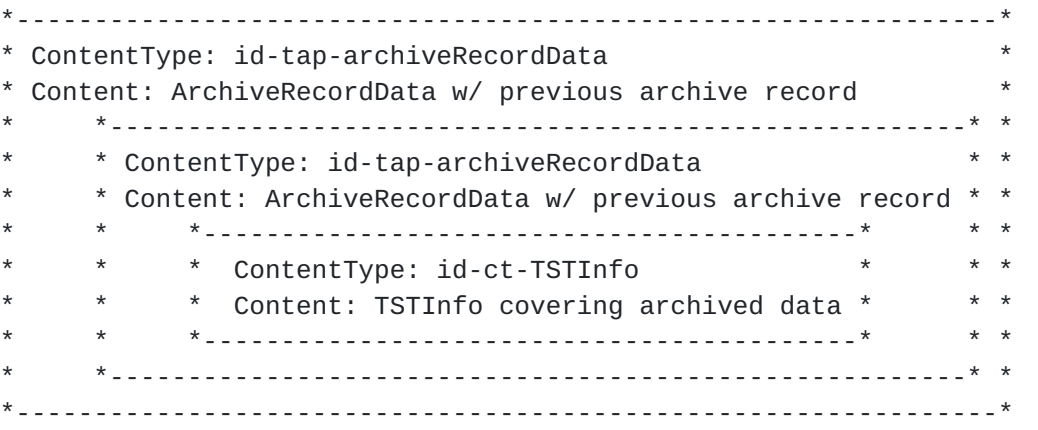

Figure 1: ArchiveRecord following two refresh operations

3.3 Deletion

Following receipt and verification of a successful ArchiveSubOrDelResp no further validation steps need be performed. However, inspection of the ArchiveControls and/or ArchiveToken returned in the response SHOULD be performed.

Deletion requests MUST be authenticated; it is RECOMMENDED that deletion requests be digitally signed in order to protect against unauthorized parties from issuing or modifying deletion requests. The deletion request client MUST perform the following processing of the deletion response in order to be compliant with this specification:

- Process each archive control per definition of control;
- Verify the signature of the TAA on the response; and
- Match the archive token returned with archive token requested.

4. Transports

There are no mandatory transport mechanisms for TAP messages. The mechanisms described below are optional.

4.1 TAP over HTTP

This section describes the formatting conventions for TAP requests and responses when carried by HTTP.

4.1.1 TAP Requests

HTTP-based TAP requests can use the POST method to submit their requests. Where confidentiality is a requirement, TAP transactions exchanged using HTTP MAY be protected using either TLS/SSL or some other lower layer protocol.

When authentication is a requirement, the request could be signed or the TAP transactions exchanged using HTTP MAY be protected using client authenticated TLS/SSL or some other lower layer protocol.

A TAP request using the POST method is constructed as follows:

The Content-Type header MUST have the value "application/tap-request".

The Content-Length header MUST be present and have the exact length of the request.

The body of the message is the binary value of the DER encoding of the request. Other HTTP headers MAY be present and MAY be ignored if not understood by the requestor.

Sample Content-Type header:

```
Content-Type: application/tap-request
```

4.1.2 TAP Response

An HTTP-based TAP response is composed of the appropriate HTTP headers, followed by the binary value of the DER encoding of the response.

The Content-Type header MUST have the value "application/tap-response".

The Content-Length header MUST be present and specify the length of the response.

Other HTTP headers MAY be present and MAY be ignored if not understood by the requestor.

5. ArchiveControls, TrackingInfos and CryptoInfos

5.1 Archive Controls

This document defines several ArchiveControls that MAY be supported by TAA implementations. Additional controls MAY also be supported. When a TAA receives a request with an unrecognized or unsupported control a response indicating failure MUST be generated and returned to the submitter of the request. Archive controls typically work in a request/response fashion, i.e. when a client includes an ArchiveControl in a request a corresponding control is expected in the response.

5.1.1 Nonce

A nonce may be included in an ArchiveControls structure using the id-tap-nonce object identifier and following ASN.1 structure:

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

A successful, associated response message MUST include an archive control with the archiveControlType field set to id-tap-nonce and the nonce from the request in the archiveControlValue field.

5.1.2 TrustAnchorRequest

As part of an ArchiveRetrievalRequest, requestors may request the set of trust anchors known to the TAA at a specific time using the id-tap-trustAnchorRequest object identifier and following ASN.1 structure:

```
TrustAnchorRequest ::= GeneralizedTime
```

A successful, associated ArchiveRetrievalResponse MUST include an archive control with the archiveControlType field set to id-tap-trustAnchorResponse and a TrustAnchorResponse in the archiveControlValue field. TrustAnchorResponse contains information about the trust anchors that were known to the TAA at the specified time. This information may be useful when verifying signatures applied to archived data.

```
TrustAnchorResponse::= SEQUENCE SIZE (0..MAX) OF TrustAnchorInfo

TrustAnchorInfo ::= CHOICE
{
   cert        Certificate,
   rawInfo     [0] RawTrustAnchorInfo
}
```

```
RawTrustAnchorInfo ::= SEQUENCE
{
   name         Name,
   algorithm    AlgorithmIdentifier,
   pubKey       BIT STRING
}
```

5.1.3 TSA Policy

The TSAPolicy archive control can be used to request that the initial timestamp obtained for an archived data submission be issued under a specific policy. A policy may be included in an ArchiveControls structure using the id-tap-tsaPolicy object identifier and following ASN.1 structure:

```
TSAPolicy ::= OBJECT IDENTIFIER
```

The TSAPolicy ArchiveControl has no response component.

5.2 TrackingInfos

This specification defines a TrackingInfo. TAA implementations are free to define tracking information objects as necessary. Clients are not required to process tracking information but SHOULD be capable of processing the TAALocation TrackingInfo.

5.2.1 TAALocation

Long-term identification of a TAA may not be practical. TAAs MAY include a TAALocation TrackingInfo to assist bearers of an archive token to locate a TAA for retrieval or deletion purposes.

TAALocations may be included in a TrackingInfos structure using the id-tap-taaLocation object identifier and following ASN.1 structure:

```
   TAALocation ::= GeneralName
```

5.3 CryptoInfos

Clients are not required to process CryptoInfos. This document defines several CryptoInfos that MAY be supported by client or TAA implementations.

5.3.1 Certificates

Certificates may be included in a CryptoInfos structure using the id-tap-certificates object identifier and following ASN.1 structure:

```
Certificates ::= SEQUENCE SIZE (1..MAX) OF Certificate
```

5.3.2 OCSPResponses

OCSPResponses may be included in a CryptoInfos structure using the id-tap-ocspResponses object identifier and following ASN.1 structure:

```
OCSPResponses ::= SEQUENCE SIZE (1..MAX) OF OCSPResponse
```

5.3.3 CRLs

CRLs may be included in a CryptoInfos structure using the id-tap-crls object identifier and following ASN.1 structure:

```
CRLs ::= SEQUENCE SIZE (1..MAX) OF CertificateList
```

6. TAP ASN.1 Module

```
PKIXTAP
-- {iso(1) identified-organization(3) dod(6) internet(1)
-- security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-tap(TBD) }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL --

IMPORTS

TimeStampToken, Accuracy
FROM
PKIXTSP {iso(1) identified-organization(3) dod(6) internet(1)
security(5) mechanisms(5) pkix(7) id-mod(0) id-mod-tsp(13) }

Name
FROM
InformationFramework { joint-iso-itu-t ds(5) module(1)
informationFramework(1) 3 }

ContentInfo
FROM
CryptographicMessageSyntax {iso(1) member-body(2) us(840)
rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0) cms(1)}
```

```
Certificate, CertificateList, AlgorithmIdentifier
FROM
AuthenticationFramework {joint-iso-itu-t ds(5) module(1)
usefulDefinitions(0) 4}

GeneralName
FROM
CertificateExtensions {joint-iso-itu-t ds(5) module(1)
certificateExtensions(26) 4}

OCSPResponse
FROM
OCSP;

--Submission transactions
ArchiveSubmissionReq ::= SEQUENCE
{
   version           TAPVersion DEFAULT v1,
   submitterName     GeneralName,
   policy            OBJECT IDENTIFIER    OPTIONAL,
   archiveControls   [0] ArchiveControls  OPTIONAL,
   archivedData      ArchivedData
}

TAPVersion ::= INTEGER { v1(0) }

ArchiveControls ::= SEQUENCE SIZE (1..MAX) OF ArchiveControl
ArchiveControl ::= SEQUENCE
{
   archiveControlType   OBJECT IDENTIFIER,
   archiveControlValue  ANY DEFINED BY archiveControlType OPTIONAL
}

ArchivedData ::= SEQUENCE
{
   type     ArchivedDataType  OPTIONAL,
   data     OCTET STRING
}

ArchivedDataType ::= CHOICE
{
   oid      OBJECT IDENTIFIER,
   mimeType UTF8String
}

ArchiveSubOrDelResp ::= SEQUENCE
{
   version           TAPVersion DEFAULT v1,
   status            ArchiveStatus,
```

```
   archiveToken      ArchiveToken         OPTIONAL,
   archiveControls   [0] ArchiveControls  OPTIONAL
}

ArchiveStatus ::= SEQUENCE
{
   code           ArchiveStatusCode,
   statusString   UTF8String  OPTIONAL
}

ArchiveStatusCode ::= ENUMERATED
{
   success              (0),-- success
   genericFailure       (1),-- misc. unspecified failure
   authenticationFailed (2),-- authentication failed (or absent)
   unauthorizedRequest  (3),-- submitter(or requestor) not authorized
   unrecognizedControl  (4),-- unrecognized or disallowed control
   controlFailure       (5),-- control processing failed
   policyFailure        (6),-- policy not supported
   timestampFailure     (7),-- timestamp could not be obtained
   retrievalDelayed     (8),-- retrieval may require manual action
   unsupportedDataFormat(9) -- format of submitted data not supported
   -- add more status codes
}

ArchiveToken ::= ContentInfo
 -- content type: id-tap-archiveToken
 -- content: ArchiveTokenData

ArchiveTokenData ::= SEQUENCE
{
   submitterName  GeneralName,
   timestamp      TimeStampToken,
   curTime        GeneralizedTime,
   trackingInfo   TrackingInfos     OPTIONAL
}

TrackingInfos ::= SEQUENCE SIZE (1..MAX) OF TrackingInfo
TrackingInfo ::= ContentInfo

--Retrieval transactions
ArchiveRetrievalReq ::= SEQUENCE
{
   version           TAPVersion DEFAULT v1,
   requestorName     GeneralName,
   retrievalRequest  ArchiveRetrievalInfo    OPTIONAL,
   archiveControls   [0] ArchiveControls     OPTIONAL
}
```

```
   ArchiveRetrievalInfo ::= CHOICE
```

```
{
   archiveToken   [0] ArchiveToken,
   archiveInfo    [1] ArchiveInfo,
   pollReference  [2] OCTET STRING
}

ArchiveInfo::= SEQUENCE
{
   tokensOnly     BOOLEAN              DEFAULT TRUE,
   submitterName  [0] GeneralName      OPTIONAL,
   timestamp      [1] TimeStampToken   OPTIONAL,
   timeInfo       [2] ArchiveTimeInfo  OPTIONAL
}

ArchiveTimeInfo ::= SEQUENCE
{
   time     GeneralizedTime,
   accuracy Accuracy    OPTIONAL
}

ArchiveRetrievalResp ::= SEQUENCE
{
   version           TAPVersion DEFAULT v1,
   status            ArchiveStatus,
   archiveControls   [0] ArchiveControls     OPTIONAL,
   results           ArchiveRetrievalResults OPTIONAL
}

ArchiveRetrievalResults ::= SEQUENCE SIZE (1..MAX) OF ArchivePackage

ArchivePackage ::= SEQUENCE
{
   archiveToken      ArchiveToken,
   packageData       [0] ArchivePackageData  OPTIONAL,
   pollReference     [1] OCTET STRING        OPTIONAL
}

ArchivePackageData ::= SEQUENCE
{
   digestAlgorithms  DigestAlgorithmIdentifiers,
   policy            OBJECT IDENTIFIER          OPTIONAL,
   archiveRecord     ArchiveRecord,
   cryptoInfos       [0] CryptoInfos            OPTIONAL,
   archivedData      ArchivedData
}

ArchiveRecord ::= ContentInfo
-- content type: id-tap-archiveRecordData
-- content: ArchiveRecordData
```

```
CryptoInfos ::= SEQUENCE SIZE (1..MAX) OF CryptoInfo
CryptoInfo ::= ContentInfo

ArchiveRecordData ::= SEQUENCE
{
   timestampedData   TimeStampedData,  -- covered by timestamp
   timestamp         TimeStampToken
}

TimeStampedData ::= SEQUENCE
{
   prevArchRecord ContentInfo,   -- previous record
   messageImprint MessageImprint -- hash of archived data
}

--Deletion transactions
ArchiveDeletionReq ::= SEQUENCE
{
   version           TAPVersion DEFAULT v1,
   requestorName     GeneralName,
   archiveToken      ArchiveToken,
   archiveControls   [0] ArchiveControls  OPTIONAL
}

-- ArchiveControls, TrackingInfos and CryptoInfos
Nonce ::= OCTET STRING

TSAPolicy ::= OBJECT IDENTIFIER

TrustAnchorRequest ::= GeneralizedTime
TrustAnchorResponse::= SEQUENCE SIZE (0..MAX) OF TrustAnchorInfo

TrustAnchorInfo ::= CHOICE
{
   cert        Certificate,
   rawInfo     [0] RawTrustAnchorInfo
}

RawTrustAnchorInfo ::= SEQUENCE
{
   name         Name,
   algorithm    AlgorithmIdentifier,
   pubKey       BIT STRING
}

-- tracking infos
TAALocation ::= GeneralName

-- crypto infos
Certificates ::= SEQUENCE SIZE (1..MAX) OF Certificate
```

```
CRLs ::= SEQUENCE SIZE (1..MAX) OF CertificateList
OCSPResponses ::= SEQUENCE SIZE (1..MAX) OF OCSPResponse
TrustAnchorInfos::= SEQUENCE SIZE (1..MAX) OF TrustAnchorInfo

-- oid categories
-- id-tap                       OBJECT IDENTIFIER ::=   {id-pkix 22}
-- id-tap-msgs                  OBJECT IDENTIFIER ::=   {id-tap 1}
-- id-tap-types                 OBJECT IDENTIFIER ::=   {id-tap 2}
-- id-tap-cryptoInfos           OBJECT IDENTIFIER ::=   {id-tap 3}
-- id-tap-controls              OBJECT IDENTIFIER ::=   {id-tap 4}
-- id-tap-trackingInfos         OBJECT IDENTIFIER ::=   {id-tap 5}

-- oids related to protocol messages
-- id-tap-archiveReq            OBJECT IDENTIFIER ::={id-tap-msgs 1}
-- id-tap-archiveSubOrDelResp   OBJECT IDENTIFIER ::={id-tap-msgs 2}
-- id-tap-archiveRetrievalReq   OBJECT IDENTIFIER ::={id-tap-msgs 3}
-- id-tap-archiveRetrievalResp  OBJECT IDENTIFIER ::={id-tap-msgs 4}
-- id-tap-archiveDeletionReq    OBJECT IDENTIFIER ::={id-tap-msgs 5}

-- extended key usage oid
-- id-kp-trustedArchive    OBJECT IDENTIFIER ::=   {id-kp 15}

-- oids for content info or attribute types
-- id-tap-archiveRecordData OBJECT IDENTIFIER::={id-tap-types 1}
-- id-tap-cryptoInfos       OBJECT IDENTIFIER::={id-tap-types 2}
-- id-tap-archiveToken      OBJECT IDENTIFIER::={id-tap-types 3}

-- oids for crypto info types
-- id-tap-certificates  OBJECT IDENTIFIER ::={id-tap-cryptoInfos 1}
-- id-tap-crls          OBJECT IDENTIFIER ::={id-tap-cryptoInfos 2}
-- id-tap-ocspResponses OBJECT IDENTIFIER ::={id-tap-cryptoInfos 3}
-- id-tap-TAInfos       OBJECT IDENTIFIER ::={id-tap-cryptoInfos 4}

-- oids for archive controls
-- id-tap-nonce              OBJECT IDENTIFIER::={id-tap-controls 1}
-- id-tap-trustAnchorRequest OBJECT IDENTIFIER::={id-tap-controls 2}
-- id-tap-tsaPolicy          OBJECT IDENTIFIER::={id-tap-controls 3}

-- oids for tracking infos
-- id-tap-taaLocation OBJECT IDENTIFIER ::= {id-tap-trackingInfos 1}

END
```

7. Security Considerations

7.1 Trust Anchors for Timestamp and Other Signature Verification on Archive Retrieval

TAAs can provide all or some of the trust anchors upon retrieval. These include all the trust anchors required to verify the various timestamps in the archive record and/or all the trust anchors known to the TAA at the time of the archive submission (i.e., the timestamp on the archived data). The latter set of trust anchors may be useful in digital signature verification on the archived data, if the data was signed.

Trust anchors provided by the TAA upon archive retrieval are transmitted securely since they are included in the signed envelope of the retrieval response. The relying party (i.e., the retrieval client) MUST use a trust anchor it trusts independent of the trust anchors provided by the TAA to verify the TAA signature on the retrieval response.

The relying party (i.e., the retrieval client) can trust the TAA provided trust anchors or can ignore them. In the latter case, only the TSA (and not the TAA) needs to be trusted for the integrity of the archived data. In other words, the relying party will be able to detect the modifications made to the archived data by the TAA. Refreshing the timestamp on the archived data before the latest (i.e., most current or outermost) timestamp expires ensures this.

7.2 Algorithm and Technology Advances

In order to protect against algorithm (i.e., hashing and digital signature) compromise and/or computing technology advances, timestamps are periodically refreshed. For each timestamp token refresh, the archived data is hashed using the latest secure hashing algorithm and a timestamp token generated using a current, secure digital signature algorithm.

7.3 Authorizations

Who is "authorized" to use the TAA is the matter of local policy. If an authorization model is implemented for any of the archive services (i.e., submission, deletion, and retrieval), the corresponding service request MUST be authenticated by the TAA in order to validate the requestor authorization.

This specification does not mandate any authorization requirements.

To claim compliance with this specification, the deletion request MUST be authenticated.

It is RECOMMENDED that a prudent local policy be established to check the authorizations for deletion requests. For example, only the submitter or authorized requestors from submitting organizations should be able to delete the data.

7.4 TSA Policy

This specification does not mandate how a timestamp under a specific TSA policy is requested. It is left as a matter of local policy. Some of the examples for requesting specific TSA policy are:

- Use of archive control (control identified by id-tap-tsaPolicy serves this purpose)
- TAA not requesting any policy
- TAA requesting specific policy based on its own requirements
- TAA mapping the TAA policy to TSA policy

7.5 Other

Data formats archived by a TAA may have requirements that relate to long-term non-repudiation beyond those identified in this specification.

TAAs should be operated with appropriate physical, procedural and personnel security controls.

TAAs must be able to obtain a trusted timestamp (either by implementing timestamp functionality or by access to a timestamp service). Timestamp-related security considerations apply (see [RFC3161]).

In support of dispute resolution, it may be desirable for TAAs to archive Certificate Policy and Certification Practice Statement documents.

It may be desirable to maintain archive data on Write Once/Read Many (WORM) media.

ArchiveControls that request server-side alteration of data, i.e. collection of certificates and CRLs, should use a response format that permits submitters to verify the timestamp contained in the archive token.

8. Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property 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; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in [RFC2028]. Copies of claims of rights made available for publication 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 Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights, which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

[RFC3161] identifies the following eight (8) United States Patents related to time stamping, listed in chronological order. This may not be an exhaustive list. Other patents MAY exist or be issued at any time. This list is provided for informational purposes; to date, the IETF has not been notified of intellectual property rights claimed in regard to any of the specification contained in this document. Should this situation change, the current status may be found at the online list of claimed rights (IETF Page of Intellectual Property Rights Notices).

Implementers of this protocol SHOULD perform their own patent search and determine whether or not any encumbrances exist on their implementation.

Users of this protocol SHOULD perform their own patent search and determine whether or not any encumbrances exist on the use of this standard.

5,001,752 Public/Key Date-Time Notary Facility
Filing date: October 13, 1989
Issued: March 19, 1991
Inventor: Addison M. Fischer

5,022,080 Electronic Notary
Filing date: April 16, 1989
Issued: June 4, 1991
Inventors: Robert T. Durst, Kevin D. Hunter

5,136,643 Public/Key Date-Time Notary Facility
Filing date: December 20, 1990
Issued: August 4, 1992
Inventor: Addison M. Fischer
Note: This is a continuation of patent # 5,001,752.)

5,136,646 Digital Document Time-Stamping with Catenate Certificate
Filing date: August 2, 1990
Issued: August 4, 1992
Inventors: Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Bell Communications Research, Inc.,

5,136,647 Method for Secure Time-Stamping of Digital Documents
Filing date: August 2, 1990
Issued: August 4, 1992
Inventors: Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Bell Communications Research, Inc.,

5,373,561 Method of Extending the Validity of a Cryptographic Certificate
Filing date: December 21, 1992
Issued: December 13, 1994
Inventors: Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Bell Communications Research, Inc.,

5,422,953 Personal Date/Time Notary Device
Filing date: May 5, 1993
Issued: June 6, 1995
Inventor: Addison M. Fischer

5,781,629 Digital Document Authentication System
Filing date: February 21, 1997
Issued: July 14, 1998
Inventor: Stuart A. Haber, Wakefield S. Stornetta Jr.
(assignee) Surety Technologies, Inc.,

Normative References

[RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[RFC2028] Bradner, S. and R. Hovey, "The Organizations Involved in the IETF Standards Process", BCP 11, RFC 2028, October 1996.

[RFC2119] Bradner, S. and , "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.

[RFC3369] Housley, R., "Cryptographic Message Syntax", RFC 3369, August 2002.

Informative References

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

Authors' Addresses

Carl Wallace
Cygnacom Solutions
7927 Jones Branch Dr. Suite 100 West
McLean, VA 22102-3305
Email: cwallace@cygnacom.com

Santosh Chokhani
Orion Security Solutions
3410 N. Buchanan Street
Arlington, VA 22207
Email: chokhani@orionsec.com

Appendix A: Support for non-TAP aware clients and alternative submission request formats

In some cases it may be desirable to accept archive submissions from clients that are not TAP-aware. The following table describes the submission alternatives.

Client software	Auth. method	Message format	Archive target
TAP-aware	Transport and/or CMS	ContentInfo containing SignedData w/ ArchiveSubmissionReq in encapContentInfo field	Contents of data field in ArchivedData structure
TAP-aware	Transport	ContentInfo containing ArchiveSubmissionReq	Contents of data field in ArchivedData structure
Non-TAP-aware	Transport and/or message format	Any other format (possibly unknown or determined from transport, i.e. mime type, file extension, etc.)	Entire message

Retrieval and deletion requests are likely to be relatively rare compared to submission requests. In the interest of supporting a broad range of submission clients, it may be desirable to support alternative archive submission formats, for example, an XML submission request. Non-TAP-compliant submission formats MUST NOT use TAP-defined transport layer type information. TAA implementations could support alternative submission types via a plug-in architecture. Regardless of submission means, archive information MUST be represented using TAP-defined archive tokens, records and packages for retrieval and deletion requests.