STANDARDS BOARD

Berne, 29 March 2006

(Agenda item 4c)

S43 Electronic PostMark (EPM) interface - Proposed update

Document by Canada Post

1 Subject Update to the UPU Technical Standard S43 Electronic PostMark (EPM) interface.	**References/Paragraphs** §§ 1 - 3
2 Decisions expected Approve the update as presented in Annex 1.	§ 4 and Annex 1

I. Background

1. This is a resubmission of the request to update the S43 standard that was rejected in the previous Standards Board meeting in Sydney. Based upon the comments and guidance of this meeting, Canada Post, leaders of the EPMSpec group of the Telematics Cooperative, has taken email dialogue with IPC and UPU Standards Programme in the attempt to clearly understand the documentation requirements and present a standard that conforms to the requirements of the Standards Board. The document in Annex 1 is the result of this dialogue.
2. The USPS has provided notice to allow the UPU to continue to use the term "Electronic PostMark" within the standard, without trademark infringement.

II. Updates to S43

3. This significant update to the standard reflects the experiences of the Telematics Cooperative members in implementing systems based upon the EPM S43-2 standard. These updates reflect the following:

a) rationalization of the PostMarkedreceipt structure to tighten up integrity, provide extensibility, and support XMLDSIG in a more symmetric way;

b) enumerations added to all elements with a finite list a values for better schema integrity;

c) automatic extension of Lifecycles for RetrieveResults and CheckIntegrity;

d) merging of the CheckIntegrity DataType with the MimeType for consistency;

e) added use case for end-to-end confidentiality;

f) vastly improved documentation and explanations right across the board;

g) LogEvent moved to the Extended category, split is now 5 core verbs 7 extended verbs;

h) notion of a re-encrypt added to Encrypt operation;

i) multiple recipient encryption lists now included;

j) version element added to all operations to better handle backward compatibility.

III. Decision expected

4. The SB is requested to approve the update to S43 as presented in Annex 1 on condition that prior to publication CEN have approved the document. If substantial changes are made the document will be referred back to the SB at its July 2006.3 meeting.

Annex:

1. S43 Electronic PostMark (EPM) interface (Draft E)

DOT.PN
GdS
17.03.2006

S43-3 Draft E

UNIVERSAL POSTAL UNION

Identification/Codification Standards

Electronic PostMark (EPM) Interface Specification

- UPU status: **1**
- Date of adoption at this status: **20 November 2003**
- Date of approval of this version: **n.a.**

Contents

FOREWORD

Postal services form part of the daily life of people all over the world. The Universal Postal Union (UPU) is the specialised institution of the United Nations that regulates the universal postal service. The postal services of its 190 member countries form the largest physical distribution network in the world. Some 5 million postal employees working in over 660 000 post offices all over the world handle an annual total of 424 billion letter-post items in the domestic service and 6 billion in the international service. Some 4,4 billion parcels are sent by post annually. Keeping pace with the changing communications market, postal administrations are increasingly using new communication and information technologies to move beyond what is traditionally regarded as their core postal business. They are meeting higher customer expectations with an expanded range of products and value-added services.

Standards are important prerequisites for effective postal operations and for interconnecting the global network. The UPU's Standards Board develops and maintains a growing number of standards to improve the exchange of postal-related information between postal operators and promotes the compatibility of UPU and international postal initiatives. It works closely with postal handling organisations, customers, suppliers and other partners, including various international organisations. The Standards Board ensures that coherent standards are developed in areas such as electronic data interchange (EDI), mail encoding, postal forms and meters.

UPU standards are drafted in accordance with the rules given in Part V of the "General information on UPU standards" and are published by the UPU International Bureau in accordance with Part VII of that publication.

This document S43–3, is the UPU equivalent of CEN/TS 15121:2004. It may be amended only after prior consultation, between CEN/TC 331 and the UPU Standards Board, in accordance with the Memorandum of Understanding between CEN and the UPU.

This Electronic PostMark (EPM) Interface specification has been developed in close relationship with the following technical standards:

- UPU Standard S33: Interoperability Framework for Postal Public Key Infrastructures. This standard is currently at Status 0.
- UPU Standard S39: Trusted Time Stamp. This standard is currently at Status 1.
- Annex A: European and International Standards Inter-relationships and Evolution, is informative.

This document is the fourth version of the specification. Substantive changes to the previous version S43–3, have been marked as changed using the Microsoft Word “Track Changes” feature and appear in an alternate color when the document is viewed from within Microsoft Word.

S43-3 Draft D itself underwent a series of interim sub-version changes as a result of implementation experiences gained while implementing the Initial Draft D sub-version 1.1. The technical highlights of this evolution from the S43-3 Draft D V1.1 and the currently tabled S43-3 Draft E sub-version 1.15 are outlined in an accompanyting document entitled “S43-3 Draft E Revision History”.

INTRODUCTION

This interface specification represents a standardized way for a postal administration or its system development teams to build an Electronic PostMark capability which they can then choose to offer to their own customers as part of their electronic service inventory.

The definition of what an EPM is and what service capability a post would have if they either built or acquired an EPM implementation that supports this standard specification is as follows.

HIGH-LEVEL EPM SERVICE DEFINITION

The EPM is essentially a digital signature verification and timestamping authority which verifies and logs as evidence, the content integrity of electronic information. The collection of technical services in an EPM can cryptographically verify and store all electronic evidence in support of potential disputes which may challenge the authenticity of events within a postal customer's automated transaction.

An EPM Service which is constructed to this specification, can support the capture and reproduction of evidence data attesting to the fact that a target business transaction was conducted and completed in an environment of integrity and trustworthiness with respect to one or more of the following attributes:

- Who originated the transaction
- Who participated in the transaction
- Were the terms, conditions, and commitments understood by all parties
- When was the document agreed to by the stakeholders, and sent to each participating party
- When was it received by each participating party
- Was the content intact throughout transmission
- Have all parties been notified of all agreed events of significance

The EPM's non-repudiation service involves selected combinations of the following key service components in order to ensure end-to-end transaction integrity and evidence collection in a confidential and auditable environment. The EPM service is a set of standardized application layer software security services aimed at facilitating the introduction and integration of these core capabilities into an target customer's business applications:

- digital signature verification
- certificate status verification
- timestamping of verified signatures (i.e. a PostMarkedReceipt)
- receipt issuance
- content timestamping
- digital signature creation
- capture of signature intent (context and user commitment)
- creation of encrypted envelopes
- decryption of encrypted envelopes
- evidence logging of all EPM Service events
- logging of user events deemed relevant to the business transaction
- tying together of EPM events into a business transaction lifecycle
- retrieval of evidence data in support of dispute resolution and future challenges in a non-repudiation

context

These services can be utilized individually or in any combination to enhance the integrity and validity associated with application events which transpire within a target customer’s automated business transaction. The process of integrating these features into an automated application is termed “EPM-enabling” the target application. Each call to the EPM Service can be looked at as a non-reputable EPM event or EPM transaction within the application’s overall business workflow. These non-repudiation events can be logically linked and tracked within an application’s business workflow to provide additional business context to an arbitrator should a challenge to the event’s authenticity be presented by any of the involved parties.

When used in EPM-prescribed and documented fashion, the EPM Service can support the following capabilities:

- non-repudiation of origin
- non-repudiation of submission
- non-repudiation of delivery
- non-repudiation of receipt

1. SCOPE

The scope of this S43-3 Draft E update is restricted to the following items:

- A substantial change in the clarity of the documentation associated with the EPM Interface Specification.
- Better coverage of the individual elements that make the request and response detail associated with EPM operational verbs.
- Associated changes to the Web Services Description Language (WSDL) and its associated XML Schema (XSD) as a result of larger participation from member Posts in the implementation of the EPM Interface Standard.

The scope of this update does NOT include:

- A description of the issues surrounding inter-operability between multiple postal EPM implementations when a business transaction lifecycle requires the participation of more than one EPM implementation in a cross-postal Administration scenario.
- Issues surrounding EPM usage in a ‘multiple Certificate Authority’ scenario where inter-operating posts are participating in a cross-border transaction as described above
- Examination of “Certificate Authority deployment model” alternatives necessitated by the cross-border scenarios described above.

2. Normative References

The following referenced documents are indispensable for the application of this document. For dated references, or references to a version number, only the edition cited applies. For undated references and where there is no reference to a version number, the latest edition of the referenced document (including any amendments) applies.

UPU Standards glossary [1]

Internet Engineering Task Force (IETF) documents

IETF Internet RFCs [2] are available at http://www.ietf.org

RFC 3161 – Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP) (August 2001), C. Adams, P. Cain, D. Pinkas, R. Zuccherato. [3] http://www.faqs.org/rfcs/rfc3161.html

RFC 3126 – Electronic Signature Formats for long term electronic signatures (September 2001), D. Pinkas, J. Ross, N. Pope [4] http://www.faqs.org/rfcs/rfc3126.html

RFC 2315 – PKCS #7 V1.5 [5] http://www.faqs.org/rfcs/rfc2315.html

RFC 2630 – Cryptographic Message Syntax (June 1999), R. Housley [5] http://www.faqs.org/rfcs/rfc2630.html

RFC 3447 – PKCS #1 RSA Cryptography Specifications V2.1 http://www.faqs.org/rfcs/rfc3447.html

RFC 3275 – XML-Signature Syntax and Processing (March 2002), D. Eastlake, J. Reagle, D. Solo [6] http://www.faqs.org/rfcs/rfc3275.html

RFC 2560 – X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP (June 1999), M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams [7] http://www.faqs.org/rfcs/rfc2560.html

RFC 2617 – HTTP Authentication: Basic and Digest Access Authentication (June 1999) Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart. http://www.faqs.org/rfcs/rfc2617.html

RFC 3280 – Internet X.509 Public Key Infrastructure Certificate and CRL Profile (supersedes RFC 2459) http://www.faqs.org/rfcs/rfc3280.html

RFC 2822 – Internet Message Format http://www.faqs.org/rfcs/rfc2822.html

Organization for the Advancement of Structured Information Standards (OASIS) [8]

SAML Core 1.1 – Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V 1.1. OASIS, November 2002 E. Maler et al. http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf

OASIS DSS - Digital Signature Services 3rd Committee Draft http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dss

OASIS DSS - Digital Signature Services - EPM Profile http://docs.oasis-open.org/dss/oasis-dss-1.0-profiles-epm-spec-cd-01.pdf

Footnotes:

[1] UPU Standards are obtainable from the UPU International Bureau, whose contact details are given in the Bibliography; the UPU Standards glossary is freely accessible on URL http://www.upu.int.

[2] Internet RFCs (Requests for Comment) are available from the Internet Engineering Task Force, c/o the Corporation for National Research Initiatives, 1895 Preston White Drive, Suite 100, RESTON, VA 20191-5434, UNITED STATES OF AMERICA. Tel: (+1 703) 620 8990, Fax: (+1 703) 620 9071, www.ietf.org

[3] References within the EPM WSDL specifications that pertain to time stamp tokens, time stamp values, or any other time stamp attributes should be assumed by any reader or implementer to be RFC 3161 compliant.

[4] This RFC is considered by most authorities to be the essential definition of what constitutes a legitimate non-repudiation service. It describes the technical criteria and pre-requisites for minimum compliance as a non-repudiation service capability. The EPM interface specification honours ES-C mandatory requirements, i.e. qualified as “required” or “shall” in the text.

[5] Defines the ASN.1 layout for all relevant PKCS objects utilised by the EPM.

[6] This RFC, commonly referred to as XMLDSIG, is the W3C’s landmark standard to which nearly all XML-based attempts to capture ASN.1 PKCS7 syntax in XML refer. The current WSDL interface allows for both PKCS7 binary ASN.1 formatting of signature objects as well as the more recent XMLDSIG signature formatting.

[7] The ValidationData element defined in the WSDL interface specification refers directly to this RFC. If an EPM implementation does not utilise an OCSP responder in its implementation, then it needs to capture the essential information described in this RFC as it pertains to ValidationData required at non-repudiation challenge time for successful evidencing. This can be accomplished through CRL evidence capture as well as signed OCSP responses, the latter being more credible. This new version of the specification provides an extensibility model whereby individual posts may implement their own ValidationData complexType to extend the abstract GenericValidationData now in the schema.

[8] OASIS (Organization for the Advancement of Structured Information Standards) is a non-profit, international consortium that drives the development, convergence, and adoption of e-business standards. The consortium produces web services standards along with standards for security, e-business, and standardisation efforts in the public sector and for application-specific markets.

3. Terms and Definitions

For the purposes of this document, the terms and definitions given in the UPU Standards glossary and the following apply.

asymmetric cryptographic algorithm

cryptographic algorithm using two related keys, a public key and a private key. One key can decrypt what has been encrypted with the other one

certificate (e.g. X509)

the public CA certified portion of a key pair in a PKI environment which binds an entity's unique name and their public key to the corresponding privately-generated private key

Certificate Revocation List (CRL)

list of revoked certificates

certification

process of creating a public key certificate binding an entity's identity to its public key

Certification Authority (CA)

entity trusted by one or more other entities to create, assign, revoke or suspend public key certificates

certification path

ordered sequence of certificates of entities which, together with the public key of the initial entity, can be processed to obtain the public key of the final entity in the path

RSA

Short for Rivest, Shamir and Adelman, RSA is an encryption algorithm developed by RSA Data Security, Inc.

Cross-certification

mechanism by which two CAs exchange certificates to implement a trusted relationship

cryptographic key

parameter controlling the operation of a cryptographic function

cryptography

discipline embodying the principles, means and methods for the transformation of data in order to hide its information content, or to prevent its undetected modification, or to prevent its unauthorized use or any combination thereof

digital signature

value, cryptographically derived from selected data using a public key algorithm, which when associated with the corresponding public key and its owner, allows a recipient of the data to authenticate its origin and verify its integrity

distinguished name

globally unique name for an entity

end entity

person, organisation, computer system or group thereof that is the subject of or uses a certificate but is not a CA or RA

NOTE An end entity is a subscriber or a relying party or both.

entity

a CA, RA or end entity

hash

one-way mathematical function that maps values from a large (possibly very large) domain into a smaller domain and that satisfies the following two properties:

- for a given output, it is computationally infeasible to find an input which maps to this output
- for a given input, it is computationally infeasible to find a second input which maps to the same output

NOTE Hashing is used to reduce a potentially long message into a “hash value” or “message digest” of fixed length, which is sufficiently compact to be inputted into a digital signature algorithm.

key

see cryptographic key

key pair

the set of keys, consisting of a public key and a private key, that are associated with an entity in a public key cryptography system

Public Key Infrastructure (PKI)

set of hardware, software, people, policies and procedures needed to create, manage, store, distribute and revoke certificates based on public key cryptography

non-repudiation

service providing proof, beyond reasonable doubt, of the integrity and origin of data which can be validated by a third party

object identifier

a sequence of integer components identifying an object such as an algorithm or attribute type

Registration Authority (RA)

entity responsible for the identification and authentication of certificate subjects but which does not sign or issue certificates

relying party

recipient of a certificate who acts in reliance on that certificate and/or on a digital signature that is verified using that certificate

RSA algorithm

a cryptographic method created by Rivest, Shamir, and Adelman for which the intellectual property rights are held by RSA Data Security.

Secure Socket Layer (SSL)

protocol developed by Netscape for encrypted transmission over TCP/IP networks

subject

an entity whose public key is certified in a public key certificate

subscriber

an end entity or subject that is considered to have an account with an EPM Service Provider providing them with the ability to subscribe to the service subject to some pre-determined contractual arrangement

Trusted Time Stamp (TTS)

record mathematically linking a data item to a time and date assured by a trusted time stamping authority

Time Stamp Authority (TSA)

trusted third party which issues and/or verifies trusted time stamps

X.509 V3 certificate extension

mechanism, defined in Version 3 of the X.509 standard, supporting the embedding of usage and policy information in a certificate

4. Symbols, Abbreviations and Acronyms

For the purposes of this document, the symbols, abbreviations and acronyms given in the UPU Standards glossary and the following apply:

CMS	Cryptographic Message Syntax (the evolution of PKCS#7)
EPM	Electronic PostMark
IETF	Internet Engineering Task Force
LDAP	Lightweight Directory Access Protocol
NA	Not Applicable.
OCSP	Online Certificate Status Protocol
PA	Postal Administration
PKCS#n:	Public Key Cryptography Standard #n (e.g. PKCS#7 or PKCS#1)
PKI	Public Key Infrastructure
RDBMS	Relational Database Management System
RFC	Internet Request For Comments
RSA	Short for Rivest, Shamir and Adelman, RSA is an encryption algorithm developed by RSA Data Security, Inc.
SAML	Security Assertion Markup Language
SNMP	Simple Network Management Protocol
SSL	Secure Socket Layer
TSA	Time Stamp Authority
TSP	Trusted Service Provider
TTP	Trusted Third Party
TTS	Trusted Time Stamp
UPU	Universal Postal Union
WSDL	Web Services Description Language
XAdES	XML Advanced Electronic Signatures
XSD	XML Schema Definition
XSLT	XML Stylesheet Language Transformation

5. Electronic PostMark (EPM) Interface Specification

5.1 EPM Core Business Services - General

This clause will describe the key services that an implementation of this specification would allow a post to support as part of formal EPM service offering to it customer base. The clause will then go on to cover the details of each operation supported by the service along with detailed descriptions of each element making up the request and response structures each operation.

NOTE This Electronic PostMark (EPM) Interface Specification describes the functionality and edit rules of the actual technical specification artifacts, which are represented by an XML Schema (XSD) and an associated Web Services Definition Language (WSDL). The current version of the both these files is contained in this document as Annexes. They can also be obtained in electronic format from the UPU Technical Standards CD-ROM or can be obtained from the UPU Standards Programme.

5.1.1 Digital Signature Verification Services

The Electronic PostMark ensures that the electronic content of all messages can be verified for both content and signer integrity as well as ensuring that all input is maintained as evidence and can be re-verified at any point in the future should a challenge be launched. Verifying digital signature integrity and certificate status is performed using PKI-based digital fingerprinting and signature verification technologies to check for both content and certificate integrity.

5.1.2 Time Stamping Services

All signature verification services are time stamped with a unique Electronic PostMark^{TM} (or EPM) attesting to the fact that the post providing the EPM Service stands behind the evidence gathered during the signing ceremony, as well as the subsequent verification status. Additionally the time at which the transaction was conducted is captured both in the Electronic Postmark's logging facility as well as within the verified signatures themselves.

5.1.3 Confidentiality Services

The Electronic PostMark offers PKI-based encryption services, which provide a high degree of confidence that sensitive business information is hidden from all but the intended recipients. Encryption at origin and decryption at destination guarantees absolute security and privacy for business transaction stakeholders.

5.1.4 Non-Repudiation Services

The Electronic PostMark retains all customer-required tracking and evidence records of significance within the business transaction life cycle.

The EPM's non-repudiation service supports non-repudiation of the following types:

- Non-repudiation of Origin
- Non-repudiation of Submission
- Non-repudiation of Delivery
- Non-repudiation of Receipt

Combined with user-authentication, timestamping, and message integrity, these tracking records ensure an extremely trustworthy end-to-end business transaction process. It is intended that the EPM Service, through the implementation of jurisdiction-specific legislative requirements, can act as a legally binding transaction notarization service both within and across Postal domains.

5.1.5 Event Logging Services

The physical storage of the evidence (i.e. escrow) data associated with EPM-logged and verified business transaction data is a core capability of the Electronic PostMarking service. These electronic records are maintained by the EPM Service provider for as many years as required by the customer and the postal service (for example the USPS maintains records for at least 7 years, the Government of

Canada has mandated that Canada Post maintain its records for a minimum of 11 years.)

5.1.6 Non-Repudiation Challenge Support Services

The Postal Administration provides the individual or organization any and all required evidence of the existence, integrity, and logged time of any business transaction tracked by the service. This information can be re-produced digitally or physically and can be sent to any required arbitrating party for their assessment.

5.2 OVERVIEW OF EPM OPERATIONS

5.2.1 General

This clause introduces the major operations (i.e. verbs) supported by the EPM Service. A short description of the functionality will be described. This will be followed by an overview of the common concepts reused across all operation within the EPM support environment.

This XML Schema describes the latest version of the EPM interface specification for the EPMService moving forward. This document represents Version 1.15 of the standard.

The EPM Service Interface specification is a digital signature platform supporting basic crypto service operations as well as a comprehensive framework for the delivery of evidentiary, witnessing, and non-repudiation services. This interface specification is dedicated to the continued support of legacy CMS/PKCS7 binary signatures. This approach allows subscribing applications to leverage the strengths of both protocols and can aid in the migration from one to the other. The schema will continue to support, in an interchangeable way, use of both CMS/PKCS7 and XMLDSIG artifacts.

The schema coverage begins by describing the details of the repetitively used Complex Types and then itemizes the core EPM SOAP operations or verbs which are supported. Each operation has three clauses: the RequestType clause, the OptionsType clause, and the ResponseType clause.

EPM implementations are free to support the "XML Signature Syntax and Processing" standard (i.e. XMLDSIG) for all elements presently carrying PKCS7 content. Selection of either format is supported across the two prevalent signature formats within this domain. XML Encryption is also supported.

The table below itemizes the updated operations supported by the EPM as well as the options that can be invoked by these operations. Both the operations and the options have been categorized according to status as either ‘core’ or ‘extended’.

To qualify as a UPU-branded EPM, the EPM implementation SHALL support the updated ‘core’ operations and options at a minimum

EPM Service Operations and Options V1.15

This subclause describes the EPM Service operations, or verbs as they are sometimes referred to, that are available to client applications. The table below provides a summary of the EPM Service interface and the available options which can be requested for each operation/verb.

Operation: / Option:	Sign	Verify	RR	CI	Post Mark	Log Event	Encrypt	Decrypt	Locate	Retrieve Summary	Retrieve Postal Attributes	Start Lifecycle
ExtendLifecycle	✓	✓			✓	✓	✓	✓	✓	✓		
EndLifecycle	✓	✓			✓	✓	✓	✓	✓	✓		
VerifyCertificate		✓					✓		✓			
StoreNonRepudiationEvidence	✓	✓	✓	✓	✓		✓	✓				
DecryptIncomingEnvelope	✓	✓		✓	✓	✓	✓					
IssuePostMarkedReceipt	✓	✓	✓	✓			✓					
ReturnSignatureInfo	✓	✓	✓				✓	✓				
ReturnX509Info	✓	✓	✓				✓	✓	✓			
EncryptResponse	✓	✓	✓		✓			✓				
ReturnTimeStampAudit			✓									

RR	–	RetrieveResults
CI	–	CheckIntegrity

▨ - UPU EPM Core

✓ - Valid Option for Operation

NOTE Postal Administrations who have upgraded their EPM service implementation to reflect newer versions of this specification (e.g. from V1.14 to V1.15) are obliged to support backward compatibility of the EPM interface specification as it applies to both processing requests/responses and honoring previously issued PostMarkedReceipts. The local Posts are free to address this backward compatibility challenge as they see fit. That is to say the EPM schema specification itself will be discreet and version specific. Posts are free to honor these discrete interface versions as they wish. The `Version` *element which is present on every request and also included in the* `PostMarkedReceipt` *can be used to help in this processing.*

5.2.2 Verify

The Verify operation is the heart of the EPM Service, since fundamentally the EPM is a Digital Signature Verification Authority. The Verify operation performs a cryptographic Verify on the incoming signature which can be in any of the following formats:

a. PKCS7/CMS SignedData ASN.1 object - ISO OID value 1 2 840 113549 1 7 2

b. PKCS7/CMS EnvelopedData ASN.1 object - ISO OID value 1 2 840 113549 1 7 3

c. XML Digital Signature and Processing (i.e. XMLDSIG) compliant XML signature

d. XML Encryption (i.e. XMLENC) compliant XML EncryptedData document

NOTE 1 It should be noted that the encrypted versions b. and d. above must actually contain a signature after decryption. The decryption is specified by simply turning on the DecryptIncomingEnvelope option. The Verify operation can optionally return a PostMarked receipt by turning on the IssuePostMarkedReceipt option. The receipt on a Verify operation is the Postal Administration's attestation of having successfully verified the signature and validated the certificate used to create that signature. This receipt, which also has a unique identifier called the TransactionKey, can be retained by the subscriber if desired. All captured and derived evidence information retained by the EPM Service.

NOTE 2 The Verify operation is invoked by the originator of the document when the originator requires Proof of Origin (i.e. non-repudiation of origin). It is often the first event in a business transaction Lifecycle. When the document is Verified and optionally PostMarked at origin, it can then can be checked by the recipient using the Verify or CheckIntegrity operation also described below.

NOTE 3 The PostMarkedReceipt carries different semantic meaning when used in conjunction with a PostMark operation where no signature is actually being verified. The attestation by the Postal Administration would be different in this case as they only may be able to attest to the existence of a particular datum at a given time as no signature verification is taking place.

5.2.3 PostMark

This operation is a superset of an elementary TimeStamp as described by RFC 3161. Additional semantic meaning is carried with the PostMark and its returned PostMarkedReceipt. This operation is normally performed as a consequence of the IssuePostMarkedReceipt option on a Verify operation. It can however, be invoked directly as well. This explicitly requested version of a PostMark, sometimes termed a Level 1 or elementary PostMark, only attests to the existence of a piece of data before a particular time. The caller can pass in several content formats to be timestamped (i.e. PostMarked), and the EPM Service will return a PostMarkedReceipt containing an RFC 3161-compliant timestamptoken as well as summary information and a signed receipt.

5.2.4 CheckIntegrity

The CheckIntegrity operation allows clients to pass in content from a previous operation (a previous Verify, PostMark, or Sign are valid operations upon which to perform a CheckIntegrity) and have that content compared against the original version stored in the EPM's non-repudiation log under the requested TransactionKey. By passing in the TransactionKey of the original operation along with the content to be checked, the EPM will validate whether that content is authentic. A common use case would be as a Proof-of-Delivery, Proof-of-Possession tool whereby a sender can be reassured that the recipient has received the signed document. This is ensured since the recipient not only signs the CheckIntegrity request but also passes in the document received from the sender. In this fashion denial of receipt by the intended recipient cannot be made. The MimeType attribute of the OriginalContentType which specifies the valid values for the type of the eContent being compared. See the description of OriginalContentType in subclause 5.4.8.

If senders require Proof of Delivery, implementers should use the CheckIntegrity operation not the Verify operation, as that is the only way to ensure that the specific document (or its hash) was actually received by the recipient. If the recipient is passing in the document on the CheckIntegrity request, as well as signing over that content See also

ClaimedIdentity in subclause 5.4.10. then the sender can enjoy Proof of Delivery and non-repudiation of receipt. This operation does not perform a cryptographic verification but rather simply compares the `OriginalContent` passed in to that which already exists within the non-repudiation database under the requested TransactionKey.

5.2.5 RetrieveResults

The RetrieveResults operation is normally reserved for challenge-time requests. However EPM implementations are free to use this operation as a 'document pickup' facility. When used in this way the "sign for pickup" facility provides end-to-end non-repudiation and proof-of-delivery without the involvement of any public eMail service provider. The PosteCS service from Canada Post and LaPoste is an example of this type of a service. See also

ClaimedIdentity in subclause 5.4.10. The qualified form of the TransactionKey including the Sequence element is required in multi-event transaction lifecycles when more than one Verify event exists within the NonRepudiation database under the selected TransactionKey. The Sequence qualifier is used to select which signature verification operation the caller wants the results for. For a list of all events in a Lifecycle, refer to the RetrieveSummary in subclause 5.6.6 operation for details.

5.2.6 LogEvent

The LogEvent operation is available to allow subscribers to log and ***system*** timestamp any content they believe is of significance to the non-repudiation life cycle. This facility is also useful when the EPM Service is participating with other system components which are supporting other events within the non-repudiation lifecycle which needs to be logged. For specific use-case examples of use of the LogEvent, please refer to subclause 5.6.4.1.

NOTE Clients wishing to have submitted data timestamped should use the PostMark operation.

5.2.7 Sign

The Sign operation is another 'special use' operation normally used in a 'Corporate Seal' scenario when a particular subscribing organization wants to sign something with an organizational or role-based identity. It should be noted that this operation, is a server-side Sign and not the Sign performed by the client endpoint (normally a customer or ISV application). It can be also used by subscribing organizations wishing to ensure their partners that they are in fact receiving content that originated from them.

NOTE *The Sign operation is not intended to be used for client-side signing. The more conventional signature-creation usage scenario is when an individual using a desktop signing application signs content which they subsequently pass to the EPM for verification and PostMarking using the Verify operation.*

5.2.8 StartLifecycle

The EPM Service supports the notion of business transaction lifecycle. Realizing that business transactions often involve multiple parties dealing with multiple documents over an extended time frame and often across country borders, the EPM Service is designed to be able to 'tie together' any number of events that are deemed 'of significance' to the participating parties.

The Startlifecycle operation can be started explicitly using this operation verb, or can be started implicitly on any other operation verb simply by initializing the TransactionKey element to a pre-existing value. The EPM Service will tie the incoming operation and all its content and results to the key specified.

The explicit version of the operation is required when the subscriber wishes to specify a ParticipatingParty list. Please refer to ParticipatingPartyType in subclause 5.4.9 for details. If the subscriber does not wish to specify ParticipatingParty entries and is willing to allow a value of 'Global' for the AccessScope element, then an explicit StartLifecycle operation is not required. This is termed implicit use of lifecycle. That is a subscriber can still start a Lifecycle by leaving the TransactionKey as null on the first (or only) operation. By default every operation is part of a lifecycle of one event.

5.2.9 Encrypt

5.2.9.1 General

This operation (together with the Decrypt) provides the native EPM's confidentiality support. The EPM Service additionally supports the ability to retrieve the public encryption certificate to be used for the encrypt operation by means of the 'CertificateID' parameter. This service is normally used by organizational subscribers wishing to encrypt content for parties they are dealing with. This service is not normally used for clients interfacing with the service from their desktops. This encryption is performed by the client desktop application.

Encryption takes place with any valid public key associated with the intended recipient.

5.2.9.2 Delegated Confidentiality Service

Another 'Special Case' usage of the EPM's confidentiality capability is the optional 'delegated confidentiality' service. This capability frees individuals from having to manage the public keys of intended recipients. When a subscriber wishes to encrypt content for confidentiality reasons, they simply encrypt that content with a public key provided to them by the Postal Administration. This single public key belongs to the post and is the only public key the customer needs to maintain on their desktop. After having encrypted the content with this post-specific public key the content is now secured for transport. This envelope can be sent to the recipient.

At the recipient's end, the recipient does not have the private key required to decrypt this envelope they just received and consequently shall ask the post's local EPM Service to Decrypt it for them. It would not make sense for the EPM to simply Decrypt the content and pass it back to the caller in the clear. So what the EPM does is encrypt the response as a result of the user turning on the EncryptResponse option on the request. The EPM requires the caller's public key in order to be able to actually encrypt the response for the caller (who is the recipient in this scenario).In order to provide the EPM with the caller's public key, the caller is obliged to sign the request. This public key is used to subsequently encrypt the response content for the caller. Please also refer to subclause 5.4.12 entitled EncryptResponse Option for more detailed coverage.

5.2.10 Decrypt

This operation (together with the Encrypt) provides the native EPM's confidentiality support. The EPM Service utilizes its own private decryption key by default and uses it for all requests which have specified the DecryptIncomingEnvelope option. Where the EPM is deployed within a subscribing organisation, it needs to be pre-configured to use customer-specific private decryption keys. Postal administrations are free to choose whether to offer this deployment model to their subscribing customer's.

EXAMPLE   Canada offers this deployment model to its subscribers.

5.2.11 Locate

This EPM Service operation supports and friendly interface to public certificate retrieval. It is useful when a client does not have the public key of a recipient they wish to encrypt content for.

5.2.12 RetrieveSummary

The RetrieveSummary operation is used to access a summary report of all the events that have taken place in a particular Lifecycle. Selected elements from each the operation are returned as an unbound repeating structure. Once the specific event within this returned list has been located, the client caller can then access that specific event's details using the RetrieveResults operation.

5.2.13 RetrievePostalAttributes

The RetrievePostalAttributes operation is used to access a list of localization attributes that are specific to a country or region's EPM service provider. This operation is used to retrieve a list of country-specific attributes by category, where each attribute is maintained as a Name/Value pair keyed by Locator and maintained in a “Yellow Pages” like directory. These attributes are used to customize the visibility of a country-issued `PostMarkedReceipt` when that receipt is viewed outside the country of receipt origin. Since the `PostMarkedReceipt` contains a `Locator` element, this element can be used to access receipt rendering detail specific to the country of receipt origin.

5.3 Common Concepts

This subclause will describe the major concepts shared across EPM operations that are not specifically associated with any individual operation. Additional non operation-related concepts is also covered

5.3.1 Authentication

The act of physically authenticating individual calls to the EPM is outside the scope of this specification. The EPM delegates all authentication mechanisms to an implementation-specific authentication facility which would normally sit out in front of the EPM. The chosen authentication mechanism can be specified and deployed by each implementing post. Every post is obliged, at a minimum, to support some level of basic or default authentication (see also `AccessLevel` = default and AccessLevel = `Signed`). This may be the Basic Authentication as specified in the HTTP protocol and supported by all Web servers. It may also be a strongly-authenticated 2-way Mutual Authentication supported by certificates. In any event all sessions with the EPM shall be SSL-based. For example, in the case of HTTP’s Basic Authentication, the *Authorization* header line sent by the client contains the username and password. The header line starting with *Authorization:* shall supply the authentication scheme used and the user name and password in the form `username:password`, as a base64 encoded string as specified in RFC 2617.

EXAMPLE   If the user name is "Aladdin" and the password is "open sesame", the header would be:

Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

In this scenario, this authentication would take place in the Web Server front-end shielding the EPM Application Server from these duties. If HTTP Basic Authentication is used as the `default` authentication mechanism, the EPM Application Server could look in the HTTP header passed to it by the front-end Web server for this line in order to grant access and establish the billing entity.

Implementations have several choices as to how to pass this authentication information back to the EPM Application Server for processing. This is the case regardless of what authentication mechanism as been used. The default is to simply rely on the application server's container technology to support access from the EPM server to the request's HTTP header. In this scenario, initialization of the `BasicAuth` element within the `ClaimedIdentity` structure by the Web server is not required. Implementations can alternatively extend the Web Server to extract the user from the Authorization line (and optionally the password) and initialize the `BasicAuth` element before it travels on to the EPM application server (see element descriptions below). In either case, the EPM will receive *an **already logged-in user identity*** which has been ***pre-cleared*** by the post's Web server. How that logged-in user string is passed back to the EPM Application Server is left to the implementing post to decide.

In this fashion postal implementations are free to authenticate users using standard approaches like HTTP Basic Authentication, or may decide to use stronger techniques involving Digest Authentication, encrypted cookies, one-time password schemes, two-factor tokens, 2-way mutual authentication using X509 certificates (2-way SSL), wsse:UsernameToken, wsse:BinarySecurityToken and accompanying dsig:Signature, or any of several other authentications schemes they chose, based on their choice and their customers' preferences. Once authenticated, the ***logged-in*** user shall be passed back to the EPM Application Server. This could be standardized and made independent of the authentication mechanism.

The `AlternateIdentity` element can also be used at the discretion of Postal implementations. This element can be for example: a SAML Assertion, a Liberty Alliance Authentication Context, an X509

Certificate, or any other identity token deemed suitable to the Postal Administration and the UPU. This element would be initialized by the appropriate authentication service which executes in front of the EPM application server. Federated Identity Management servers could work in this fashion as well. This is consistent with the ***delegated authentication*** approach used by the EPM, and provides the most flexibility and choice for implementing Posts.

The `BasicAuthType` below can optionally be used for the default authentication scenario described above. It represents Basic Authentication as would be supported in HTTP Server Authentication using the "Authorization: Basic" HTTP Header line. This is but one of several authentication schemes that may be employed. The `AlternateIdentity` element is included for generic support for other schemes (e.g. SAML, Liberty Alliance, other federated identity schemes, etc ...)

```
<xs:complexType name="BasicAuthType">
    <xs:sequence>
        <xs:element name="UserID" type="xs:string"/>
        <xs:element name="Password" type="xs:string" nillable="true"/>
    </xs:sequence>
</xs:complexType>
```

NOTE   These elements only travel between the Web Server and the Application Server. These elements are initialized by the calling client and passed up as part of the HTTP Header where the Web server will perform the authentication.

Once logged in, the EPM Application Server only needs to know the identity or UserID of the requester who has already been authenticated by the Web Server normally for account management related processing or Lifecycle checks. The Password has only been included for potential future use in unforeseen authentication scenarios and should not be required by the EPM Application Server since authentication has already taken place in the Web front-end. As such it is marked as nillable. See also

ClaimedIdentity in subclause 5.4.10.

5.3.2 Transaction Handling

Every request/response within the EPM Service is assigned a unique transaction identifier called the TransactionKey. The TransactionKey is made up of three sub-elements making up the entire composite key. The Locator, the Key, and the Sequence. The Locator identifies an instance of the entire EPM Service. Usually there is one per Postal Administration, although there may be more for numerous operational and jurisdictional reasons. The Key is the unique identifier and is generated by the EPM Service when a new Lifecycle is created. This is the default (i.e. that every transaction is a single event in a single business Lifecycle. This transaction handling is implicit, and the caller need do nothing special if multi-event Lifecycles are not required. As one can see, transactions within the EPM Service are logical transactions. Each atomic operation is completely logged to the database as part of a single DBMS logical unit of work. When multiple events need to be tied together, this is termed a Lifecycle, or a multi-event Lifecycle. This is the subject of the next subclause.

5.3.3 Lifecycle Management

The EPM Service supports the notion of business transaction lifecycle. Realizing that business transactions often involve multiple parties dealing with multiple documents over an extended time frame and often across country borders, the EPM Service is designed to be able to ‘tie together’ any number of events that are deemed ‘of significance’ to the participating parties.

A lifecycle can be started explicitly using the StartLifecycle operation, or can be started implicitly on any other operation verb simply by initializing the TransactionKey element to a pre-existing value and turning on the ExtendLifecycle option. The EPM Service will tie the incoming operation and all its content and results to the key specified.

NOTE   In order to extend a Lifecycle beyond a single event, one shall always set the `ExtendLifecycle` flag as well as specifying the `TransactionKey` of the Lifecycle to which this event should be added.

Their exists an explicit mechanism for Lifecycle management whereby a StartLifecycle operation is used and provides a way for the caller to specify a `ParticipatingParty` list. Please refer to ParticipatingPartyType in subclause 5.4.9 for details. If the subscriber does not wish to specify `ParticipatingParty` entries and is willing to allow a value of `Global` for the `AccessScope` element, then an explicit StartLifecycle operation is not required. This is termed implicit use of lifecycle whereby Lifecycles are extended by setting the `TransactionKey` to a known value. That is, a subscriber can still start a Lifecycle by leaving the `TransactionKey` as null on the first (or only) operation. Subsequent events can be added to this Lifecycle by supplying the same `TransactionKey` on subsequent calls. By default every operation is part of a Lifecycle of one event.

The `ParticipatingParty` complex element (refer to ParticipatingPartyType in subclause 5.4.9 for details) represents the restricted groups or individuals who are given access to the transaction contents of a given lifecycle. `AccessScopes`, will be referenced when validating `AccessLevel` and permissions for this lifecycle and its contents.

5.3.4 Error Handling

Error handling within the EPM Service has both a terse and verbose version. Applications wishing to test overall outcome in a simple pass/fail sense can interrogate the `TransactionStatus` element. Its return value is simply: 0 – success, 1 – warning, or #### – an error number relating to the specific error encountered. Additional details are carried in the more verbose `TransactionStatusDetail` element. Its sub-elements are broken down as per TransactionStatus and TransactionStatusDetailType in subclause 5.4.1.

5.3.5 Processing Directives or Options

Every EPM Service has an Options structure where the caller can specify the special handling directives they would like to have performed for their particular request. Each Options structure contains only the options that are valid for that operation. The options which are valid for a given operation are enumerated in the RequestOptions subclause of every verb. Additionally the schema reflects only the `ValidOptions`.

5.3.6 PostMarking

A PostMarkedReceipt is a superset of a standard timestamptoken. PostMarking can be viewed as attestation to the existence of datum in time as well as the integrity of its content. The integrity of its content is the assurance that the data has not been modified since it was PostMarked. The PostMark can also serve as attestation of a successful signature verification by logging all significant non-repudiation events in a business transaction’s lifecycle.

Operations that support the `IssuePostMarkedReceipt` element perform an implicit PostMark and return a `PostMarkedReceipt`. If you specify `IssuePostMarkedReceipt` in the RequestType element for that operation, you will receive the `PostMarkedReceipt` in the operation’s response.

5.4 COMMON SCHEMA TYPES USED ACROSS EPM OPERATIONS

The following object types represent common WSDL element types that are sent to and returned from the EPM Service. These common schema complexTypes are used by most operations. Complex types contain a set of child elements that may reference other common complex or simple types.

5.4.1 TransactionStatus and TransactionStatusDetailType

For consistent error response handling, EPM-enabled applications are expected to check the overall operation status returned in the `TransactionStatus` element. It should contain a value of 0 for success or a specific error number related to the error encountered. More detailed error information is optionally returned in the `TransactionStatusDetail` complex element. For instance, if a database error occurs while writing to the non-repudiation log, the `TransactionStatus` might contain 5000 and the `ErrorNumber` could contain for example the ORA- specific error number, and the `ErrorMessage` could contain the Oracle specific text which accompanies the ORA- error number. The following are the Transaction Status related elements:

```
<xs:element name="TransactionStatus" type="xs:string"/>
<xs:element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
<xs:complexType name="TransactionStatusDetailType">
   <xs:sequence>
      <xs:element name="ErrorNumber" type="xs:string"/>
      <xs:element name="ErrorMessage" type="xs:string" nillable="true"/>
   </xs:sequence>
</xs:complexType>
```

NOTE For the benefit of independent client handling, cross-border scenarios, and ISV applications, a companion document which standardizes on handleable EPM ErrorNumber's and their associated ranges exists. It is titled Electronic PostMark Standardized Error Numbering Scheme V1.0. Please consult the local Postal Administration for details and availability.

5.4.2 TransactionKeyType

This complex type contains three elements that make up the identifier for the specific events in a lifecycle. When users or applications are adding another event to an existing lifecycle, they need only supply the Key portion of the type. When they are referring to a particular event within the lifecycle, as is the case with the CheckIntegrity operation, then both the Key and the Sequence are required. See Lifecycle Management in subclause 5.3.3 for more information. The Key is a unique identifier to that transaction. By default a TransactionKeyType complex element is populated and returned by the EPM Service. For targeted data retrieval functions such as CheckIntegrity, RetrieveResults, and selected RetrieveSummary scenarios, one shall provide the `Locator`, `Key`, and `Sequence` elements.

```
   <xs:complexType name="TransactionKeyType">
      <xs:sequence>
         <xs:element name="Locator" type="epm:LocatorType"/>
         <xs:element name="Key" type="xs:string"/>
         <xs:element name="Sequence" type="xs:string" nillable="true"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="LocatorType">
      <xs:sequence>
         <xs:element name="CountryCode" type="xs:string"/>
         <xs:element name="Version" type="xs:string"/>
         <xs:element name="ServiceProvider" type="xs:string" nillable="true"/>
         <xs:element name="Environment" type="xs:string" nillable="true"/>
      </xs:sequence>
   </xs:complexType>
```

Descriptions of the TransactionKey elements follow:

Locator – A complex type element representing a unique EPM Service instance identifier, usually the post's 2 character country code, as per ISO 3166 enumerated at

http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html#af

and the `Version` number of the EPM Interface Specification (e.g. 1.15). This element is returned by the EPM as part of the `TransactionKey` identifier. Caller's do not use this element to point at a particular EPM instance, that is accomplished through the EPM's SOAP URL identifier. The `Locator` plays a role in cross-border EPM transactions involving more than one post or country. Since the `CountryCode` is part of the `TransactionKey`, any PostMarked document can be traced back to the country that issued the PostMark. If the post has delegated EPM hosting and facilities management to more than one service provider, then that `ServiceProvider` name may be included in the `Locator` complex type. The `Environment` element allows support for more than one instance of the EPM Service for posts who wish to exploit this possibility (e.g. Production, Pilot, Training, etc …) and qualifies the transaction. These sub-elements do nor replace the need for discrete URLs for the specific EPM Service instance that callers wish to access.

Key – A string element representing the unique transaction identifier. This `Key` is originally generated by the EPM Service and returned as part of the Transaction Key identifier. This Transaction Key provides the EPM Service with a mechanism that allows later retrieval of non-repudiation evidence for a particular transaction. This element is set to null in the operation request unless: the caller is performing a RetrieveResults, RetrieveSummary, or CheckIntegrity operation in which the target Key shall be identified for retrieval, or the caller wishes to extend the transaction lifecycle (See Lifecycle Management in subclause 5.3.3 for more information).

Sequence – A string element uniquely identifying a particular event within a multi-event business transaction lifecycle. Their exist a 'sequence' of operations within an extended business transaction that involves multiple calls to the EPM Service using the same `Locator` and `Key`. Each time the same key is used, the current sequence number is incremented by one, such that all the events within the business transaction lifecycle are grouped under the same identifier (i.e. the first two elements of the composite key: `Locator` and `Key`). Only the `Sequence` number varies in these transactions. By default, calls to the EPM are not part of an extended business transaction lifecycle, and so they are assigned a unique `TransactionKey` and always contain a sequence of '1'. The `Sequence` element is always returned as part of the complex `TransactionKey` identifier. The `Sequence` can be set to null in the operation request, except when performing a RetrieveResult, RetrieveSummary, or a CheckIntegrity.

5.4.3 QualifiedDataType

The QualifiedDataType is used throughout the EPM schema. It represents a base64Binary Data stream qualified by a particular MimeType value. Please refer to the specific context and usage wherever `MimeType` is used for verb-specific details. Examples include: `application/octet-stream, text/plain, text/xml, application/pkcs7-signature`, etc … For responses, if the EPM is unable to determine the `MimeType` of the associated element, this attribute will not be present.

```
<xs:complexType name="QualifiedDataType">
   <xs:simpleContent>
      <xs:extension base="xs:base64Binary">
         <xs:attribute name="MimeType" type="xs:string" use="optional"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
```

5.4.4 SignatureInfoType

This complex type contains several sub-elements reflecting the outcome of the signature creation/verification or encryption/decryption operation. This group of elements represent a further detailed breakdown of the PKCS7 object acted upon as part of the operation, including the original content or data that the operation was performed on. A SignatureInfoType complex element is populated and returned by the EPM Service. The following are the elements it contains. It should be noted that the information associated with these elements is already present in XMLDSIG-based signatures and use of this structure becomes academic.

SignedContent – A base64Binary encoded response element representing the original content when either a Sign, Verify, CheckIntegrity, RetrieveResults, or Encrypt (special case) is performed. For example on a Verify of an incoming PKCS7 enveloping signature, the content that was originally signed will be returned in the `SignedContent` element.

ContentHash– A string element representing the result of the one-way hash performed using the specified hash algorithm. The hash is applied over the content to be signed. This value will be null for encrypt or decrypt operations.

ContentHashAlgo – A string element representing the algorithm used to produce the one-way hash value resulting from the signature creation process. This element is returned by the EPM implementation for informational purposes only and will not affect inter-operability. Supported values are standardized but remain EPM implementation specific. The value `sha1WithRSAEncryption` is the default PKCS1 signature creation algorithm and **SHALL** be supported by EPM implementations when creating and verifying signatures. This signature algorithm is also the default for cross-border signatures sent between the EPMs of other Postal Administrations. The `md5WithRSAEncryption` algorithm **SHOULD** also be supported by posts for PEM and S/MIME compatibility. Individual Posts may introduce additional signature algorithms as required. Examples might include: `sha256WithRSAEncryption` or `sha512WithRSAEncryption`. Please refer to RFC 2315 PKCS #7 V1.5 and the evolutionary series of standards starting with RFC 2313, RFC 2437, and RFC 3447 covering PKCS #1.

XMLDSIG-based signatures are self-documenting in this regard.

ContentEncryptAlgo – A string element representing the name of the algorithm used to perform encryption or decryption of the content. This element is also EPM implementation specific. EPM implementations **SHALL** support tripledes-cbc with RSA Key Transport as the default algorithm. This value will be null except for encrypt or decrypt operations. See **SessionKeyAlgo** in the Encrypt operation for further details on algorithm use.

SigningTime – A string element representing the time that the signature operation was performed. It is extracted from the signature object. If a client application originally created the signature and is asking the EPM Service to Verify it, this element will contain the signing time as created by the client application. This value is provided in the standard UTC Format – YYYYMMDDHHMMSS plus the character 'Z'.

PKCS1 –The SignatureValue from within the PKCS7 signature. Does not apply for XMLDSIG-based signatures where the `SignatureValue` element plays that role. Returned as a courtesy function to relieve callers from having to parse ASN.1 binary signatures.

```
   <xs:complexType name="SignatureInfoType">
      <xs:sequence>
         <xs:element name="SignedContent" type="epm:QualifiedDataType"
nillable="true"/>
         <xs:element name="ContentHash" type="xs:string" nillable="true"/>
         <xs:element name="ContentHashAlgo" type="xs:string" nillable="true"/>
         <xs:element name="ContentEncryptAlgo" type="xs:string" nillable="true"/>
         <xs:element name="SigningTime" type="xs:string" nillable="true"/>
         <xs:element name="PKCS1" type="epm:QualifiedDataType" nillable="true"/>
      </xs:sequence>
   </xs:complexType>
```

5.4.5 X509InfoType

This complex element contains several sub-elements reflecting information from the certificate used in the particular EPM operation performed. It contains a concatenation of signing or encryption certificate information. It will also contains information on any applicable certificate revocation status information.

NOTE *The revocation attributes of the X509Info structure will be null if the either the certificate was not revoked or the 'VerifyCertificate' option was not selected.*

A X509InfoType complex element is populated and returned by the EPM Service. The following are the elements it contains:

X509Subject – A string element representing the fully qualified Distinguished Name of the certificate holder.

X509Issuer – A string element representing the Certificate Authority (CA) that issued and signed the certificate.

X509Serial – A string element representing a unique serial number to identify the certificate issued by the Certificate Authority.

X509StatusSource – A string element representing the name of the source used to validate that the certificate has not been revoked. Valid values are: ‘CRL’ or ‘OCSP’.

X509ValidFrom – A string element representing the issue date of the certificate.

X509ValidTo – A string element representing the expiration date of the certificate.

X509Certificate – A base64-encoded public certificate.

X509RevocationReason – A string element representing the reason ‘code’ for the certificate revocation. This value will be null unless a revoked user certificate is detected during a Verify, Locate, or Encrypt operation in which the ‘VerifyCertificate’ option was set or defaulted to true. Valid values should be those described in RFCs 2459 and 3280. Codes available only for Version 2 CRLs.

X509RevocationReasonString – A string element representing a description of the reason code for the certificate revocation. This value will be null unless a revoked user certificate is detected during a Verify or Encrypt operation in which the VerifyCertificate option was set to true. Valid values should be those described in RFCs 2459 and 3280. Codes available only for Version 2 CRLs.

X509RevocationTime – A string element representing the time the signature was revoked by the Certificate Authority. This value will be null unless a revoked user certificate is detected during a Verify or Encrypt operation in which the VerifyCertificate option was set to true.

GenericValidationData – This element is post-specific and represents the evidence provided by the EPM which attests to the fact that the certificate status has been checked. As such it is an abstract type which can be implemented as the Posts see fit. A default implementation called `X509ValidationData` is included representing a `QualifiedDataType` used to hold an RFC 2560 OCSP signed response. This default implementation represents a binary value element holding a binary-encoded OCSP response signature from the provider of the certificate validation information. This encoded element contains the OCSP Validation Data returned. It is signed content as per RFC 2560 and also described in RFC 3126 as would be returned by a standardized OCSP Responder.

If an EPM implementation is not using an OCSP responder, then that post is free to redefine their own implementation of the abstract type. Sufficient certificate chain and revocation references should be included here as would be mandated by local signature laws. Additionally, many jurisdictions (e.g. the EU) require that this validation info be signed by the trusted service provider. This is not a problem when using an RFC 2560-compliant OCSP. The default implementation contains a MimeType attribute `QualifiedDataType` which will contain valid values of either `text/xml` when the `X509ValidationData` is an XMLDSIG-formatted signature, or `application/pkcs7-signature`, when it is an ASN.1 binary PKCS7 signature.

```
<xs:complexType name="X509InfoType">
   <xs:sequence>
      <xs:element name="X509Subject" type="xs:string"/>
      <xs:element name="X509Issuer" type="xs:string" nillable="true"/>
      <xs:element name="X509Serial" type="xs:string" nillable="true"/>
      <xs:element name="X509StatusSource" type="xs:string"/>
      <xs:element name="X509ValidFrom" type="xs:string"/>
      <xs:element name="X509ValidTo" type="xs:string"/>
      <xs:element name="X509Certificate" type="xs:string" nillable="true"/>
      <xs:element name="X509RevocationReason" type="xs:string" nillable="true"/>
      <xs:element name="X509RevocationReasonString" type="xs:string"
nillable="true"/>
      <xs:element name="X509RevocationTime" type="xs:string" nillable="true"/>
      <xs:element name="X509ValidationData" type="epm:X509ValidationDataType"
```

```
nillable="true"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="GenericValidationDataType" abstract="true">
   <xs:sequence>
      <xs:element name="GenericValidationData" type="xs:anyType"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="X509ValidationDataType">
   <xs:complexContent>
      <xs:extension base="epm:GenericValidationDataType">
         <xs:sequence>
            <xs:element name="X509ValidationData" type="epm:QualifiedDataType"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

5.4.6 PostMarkedReceipt

The PostMarkedReceipt is returned when either the `IssuePostMarkedReceipt` option has been specified or if a direct PostMark operation has been requested over user-supplied content. When the `IssuePostMarkedReceipt` element is included on a Verify operation for example, the returned receipt is the Postal Administration's attestation of having successfully performed the requested operation. In the case of a Verify for example, this would be an attestation of having successfully verified the signature and the status of the certificate used to create it. When the `IssuePostMarkedReceipt` element is included on a Sign operation, the returned receipt is the Postal Administration's attestation of having created this signature with the desired signing key and that this key/certificate is valid for its intended purpose. This receipting mechanism is at the heart of the EPM's non-repudiation capability. The `PostMarkedReceipt` contains three core pieces of information, the Receipt, a TimeStampToken, and a signature of authenticity binding everything together. The XML layout of the `PostMarkedReceipt` reflects the signature type used to produced it. That is, when a `PostMarkedReceipt` is returned in response to a PKCS7-based operation, the layout is made up of a Receipt structure containing basic receipt information, optional receipt metadata, and a TimeStampToken, as well as a detached signature to ensure authenticity. When the `PostMarkedReceipt` is in response to an XMLDSIG-based operation, the XML digital signature, whose `Reference's` cover the same information, itself represents the `PostMarkedReceipt.` That is to say, the signature is the `PostMarkedReceipt.`

```
<xs:complexType name="PostMarkedReceiptType">
   <xs:sequence>
      <xs:choice>
         <xs:element name="PKCS7SignedReceipt" type="epm:PKCS7SignedReceiptType"/>
         <xs:element name="XMLSignedReceipt" type="dsig:Signature"/>
      </xs:choice>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="PKCS7SignedReceiptType">
   <xs:sequence>
      <xs:element name="Receipt" type="epm:ReceiptType"/>
      <xs:element name="ReceiptSignature" type="epm:QualifiedDataType"
nillable="true"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="ReceiptType">
   <xs:sequence>
```

```
      <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
      <xs:element name="Requester" type="xs:string"/>
      <xs:element name="Operation" type="xs:string"/>
      <xs:element name="TSAX509SubjectName" type="xs:string"/>
      <xs:element name="TimeStampValue" type="xs:string"/>
      <xs:element name="RevocationStatusQualifier" type="xs:string"/>
      <xs:element name="TimeStampToken" type="epm:QualifiedDataType" nillable="true"
minOccurs="0" maxOccurs="1"/>
      <xs:element name="MessageImprint" type="xs:base64Binary" nillable="true"/>
      <xs:element name="PostMarkImage" type="epm:QualifiedDataType" nillable="true"/>
      <xs:element name="ReceiptMetadata" type="epm:ReceiptMetadataType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="ReceiptMetadataType">
   <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:choice>
         <xs:element name="Value" type="xs:string"/>
         <xs:element name="EncodedValue" type="epm:QualifiedDataType"/>
      <xs:choice>
   </xs:sequence>
</xs:complexType>

<xs:complexType> name="IssuePostMarkedReceiptType">
   <xs:sequence>
      <xs:element name="Location" type="epm:ValidLocation" minOccurs="0"/>
      <xs:element name="PostMarkImage" type="epm:PostMarkImageType" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="PostMarkImageType">
   <xs:simpleContent>
      <xs:extension base="xs:boolean">
         <xs:attribute name="Format" type="xs:string" default="JPG"/>
         <xs:attribute name="Size" type="epm:ValidImageSize" default="Small"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
```

TimeStampToken – This is the RFC 3161 timestamptoken taken over the appropriate content as dictated by the context in which the `PostMarkedReceipt` was requested. For example, in the case of a `PostMarkedReceipt` requested on a `Verify` or a `Sign` operation via the `IssuePostMarkedReceipt` option, the timestamptoken will be a "signature" timestamp taken over the signature value of the signature just verified (Verify) or created (Sign). In the case of direct invocation of a PostMark operation, the timestamp is calculated over the content passed in on the request. The PostMark operation supports several content types which can be timestamped.

For PKCS7-based signatures, as specified by the `SignatureType` on the PostMark operation or the signature type being verified in the case of a `Verify` operation, the `PostMarkedReceipt` is returned as a standalone receipt. The `IssuePostMarkedReceipt` element shall be included in the request in order to obtain a receipt.

`TimeStampToken` is the actual timestamp formatted as a binary PKCS7 ASN1 signature. The default signature creation algorithm for the EPM is `sha1WithRSAEncryption` and **SHALL** be supported by EPM implementations when creating and verifying signatures of all types including the `TimeStampToken` described in this subclause.

NOTE The value of the `MimeType` attribute of this element is the MimeType of the content that was *timestamped. For example if a binary file was timestamped, then the MimeType attribute of the TimeStampToken will be `application/octet-stream`, and if the content being timestamped was a signature, then the MimeType of the TimeStampToken will be `application/pkcs7-signature`.*

When a `PostMarkedReceipt` is issued in response to a Verify the `PostMarkedReceipt` will be either `standalone` or `embedded` into the incoming signature being verified by the EPM. This is determined by the value of the `Location` sub-element of the `IssuePostMarkedReceipt`.

The effect of the `Location` sub-element of the `IssuePostMarkedReceipt` option is as follows:

- For `IssuePostMarkedReceipt` on a **Verify** operation of an XMLDSIG signature
    - If the `Location` sub-element is specified as `standalone`, a standalone `PostMarkedReceipt` structure will be returned in the Verify response.
    - If the `Location` sub-element is specified as `embedded`, the `PostMarkedReceipt` structure will be included or embedded in the signed document being verified and will be returned in the `SignatureData` element of the Verify response.
- For `IssuePostMarkedReceipt` on a **Verify** operation of a CMS/PKCS7 signature
    - If the `Location` sub-element is specified as `standalone`, a standalone `PostMarkedReceipt` structure will be returned in the Verify response. This is consistent with the XMLDSIG handling above.
    - If the `Location` sub-element is specified as `embedded`, the `PostMarkedReceipt` structure will be returned as above, but additionally the RFC 3161 compliant timestamptoken will also be `embedded` in the signature being verified as an unauthenticated attribute and will be returned in the `SignatureData` element of the Verify response.

When a `PostMarkedReceipt` is issued in response to a PostMark operation whose primary intent is to PostMark data or content, and if the SignatureType is XMLDSIG, the `PostMarkedReceipt` will reference the incoming XML content via a signature reference whose `object` is the data being PostMarked and timestamped. The content and the `PostMarkedReceipt` will be returned together as part of the `PostMarkedReceipt` structure.

See Annex B (Informative) Examples for a description of layouts and references of both types. Please also refer to the `PostMark` operation in subclause 5.5.2 for more details on supported content types.

ReceiptSignature – This element contains a detached signature over the contents of the Receipt. The ReceiptSignature is only used for receipts issued in response to PKCS7-based operations. It is an EPM-produced detached PKCS7 signature over the Receipt structure. It is used simply to protect the authenticity of this standalone XML-formatted receipt information. It is not required with XMLDSIG-based receipts whose references are protected by the PostMarkedReceipt signature itself. An example of the required eContent which is input to the PKCS7 ReceiptSignature creation follows. The same serialization rules described under the paragraph title Serialization Conventions in subclause 5.4.10 apply here as well.

*NOTE The default signature creation algorithm for the EPM is `sha1WithRSAEncryption` and **SHALL** be supported by EPM implementations when creating and verifying signatures of all types including the `ReceiptSignature` described above.*

```
<Receipt>
<TransactionKey>
<Locator>
<CountryCode>CA</CountryCode>
<Version>114</Version>
<ServiceProvider>1</ServiceProvider>
<Environment></Environment>
</Locator>
<Key>041019-133230-59841915</Key>
```

```
<Sequence>1</Sequence>
</TransactionKey>
<Requester>.....</Requester>
<RevocationStatusQualifier></RevocationStatusQualifier>
<TimeStampToken MimeType="application/pkcs7-signature">.....</TimeStampToken>
<ReceiptMetadata>
<Name>TimeStampValue</Name>
<Value>2004-03-27T17:47:18.750</Value>
</ReceiptMetadata>
<ReceiptMetadata>
<Name>.....</Name>
<Value>…..</Value>
</ReceiptMetadata>
</Receipt>
```

Receipt –The `ReceiptType` element below is a sub-element of the `PostMarkedReceipt` and along with the `TimeStampToken` and `ReceiptSignature` make up the `PostMarkedReceipt` for PKCS7-based receipts. The Receipt links the timestamp to the TransactionKey as well as providing additional summary information.

TransactionKey – The TransactionKey is included as part of the Receipt to allow client applications to locate the source of the receipt, which is very important is cross-border transactions.

Requester – The element should contain the authenticated user. Its value should reflect the chosen authentication method used by each particular post. Thus if a post is using strong 2-way X509 Mutual Authentication, then this element could contain the X509SubjectName (i.e. the DN). If another post is using delegated signing and employs One-Time-Passwords (OTPs) for example, this element could be the pre-registered UserID associated with the OTP token. If another post is using 1-way SSL Basic Authentication, this element would contain the UserID registered to the post's front-end Web Server.

Operation – The EPM operation or verb upon which the `PostMarkedReceipt` was issued e.g. Verify, CheckIntegrity, RetrieveResults, etc.

TSAX509SubjectName –The TSAX509SubjectName (i.e. the DN) element is included for convenience so users need not parse the PKCS7 for it. It is redundant for XMLDSIG based TimeStampTokens as illustrated in Annex B (Informative) Examples.

TimeStampValue –The TimeStampValue is the UTC time exactly as returned by the TSA in UTC Z (Zulu) format as per RFC 3161. An example of a string in this format would be laid out as follows: YYYYMMDDHHMMSS plus the character 'Z'.

RevocationStatusQualifier – The RevocationStatusQualifier will reflect whether CRL checking was performed or not. Valid values are `Checked`, `Not Checked`, and `Not Applicable`. The `Not Applicable` value is required when this PostMarkedReceipt was returned as a result of a direct PostMark operation over user-defined content. In this case no revocation checking of any sort is performed. The PostMark is just attesting to the existence of that data before this point in time.

MessageImprint – This element is a copy of the messageImprint field from the TimeStampToken's `TstInfo` structure. It has been duplicated in the `Receipt` structure to facilitate verification (especially on client desktop applications) since it alleviates the verifier from having to ASN.1 parse the `TimeStampToken` in order to extract the `messageImprint` field containing the hash to be compared. EPM implementations are obliged to initialize this element. Please also refer to the discussion dealing with the verification of the `PostMarkedReceipt` in subclause 5.5.1.3 under the PostMarkedReceipt topic. It is not required for XMLDSIG-based PostMarkedReceipts.

PostMarkImage – This is an optional element which can contain a post-specific image reflecting the PostMark issued for this operation. It would be generated by the EPM implementation and returned as part of the `PostMarkedReceipt`. Client applications could display this element to the caller in desktop application usage scenarios. This element could also be useful for integration with ISV desktop applications or post-specific desktop and browser applications. Please also refer to the Location sub-element of the `IssuePostMarkedReceipt` where `PostMarkImageSize` and `PostMarkImageType` can be used to instruct the EPM to create a `PostMarkImage` and return it in this response element.

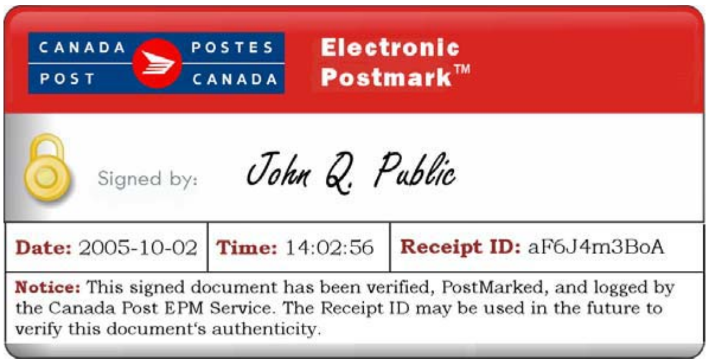
ReceiptMetadata – This optional unbound element provides a vehicle for Posts to add any country-specific textual or binary evidentiary content to be included in the `PostMarkedReceipt`. It is covered by the `ReceiptSignature` and therefore shall be prepared on the EPM server when the PostMark is initially created. Through the element choice construct, base64-encoded data, intended for use with binary artifacts such as documents, additional signatures, audit signatures, etc … can also be included. Posts for example, could use this element to include the document and the signature that was signed and PostMarked, thus allowing the PostMarkedReceipt to act as a single container for all relevant non-repudiation information. It could also be used to hold an OCSP signed response or XAdES timestamps as well. Posts are free to use this `ReceiptMetadata` element as they wish. Its intended use is for additional evidentiary and attestation information a post may wish to add to the `PostMarkedReceipt`.

IssuePostMarkedReceipt – This element is included here for reference and does not appear in the `PostMarkedReceipt` structure itself but rather appears in the RequestOptions subclause of the EPM operations which support the `PostMarkedReceipt` i.e. Sign, Verify, RetrieveResults, CheckIntegrity, and Encrypt. The presence of this option instructs the EPM to generate a `PostMarkedReceipt` and include it in the response. This element as 3 sub-elements described below.

Location – This optional element instructs the EPM where to place the resultant `PostMarkedReceipt`. Valid values are `standalone` and `embedded`. The default if omitted is `standalone`. A value of `standalone` instruct the EPM to return the `PostMarkedReceipt` in its own response element distinct from the signature. For Sign and Verify operations a value of `embedded` instructs the EPM to embed the resultant `PostMarkedReceipt` in the signature just created or verified.

PostMarkImage – This optional boolean element when set to `true` instructs the EPM to return an image associated with the `PostMarkedReceipt`. This image can be a fixed image set by the post, or it can be dynamically rendered to contain a Postal logo as well as selected PostMark information such as the `Signator` and/or the `TimeStampValue`. It however needs to be a displayable image. It has 2 attributes `Format` and `Size`. The example below illustrates a `PostMarkImage` that has 4 pieces of dynamic information painted on top of the image background and rendered as a JPG. This response element alleviates the client or ISV application from having to render an image to represent that the document has been PostMarked.

NOTE This image is not to be confused with the Universal EPM Logo which a simply a non-PostMark-specific, non-Transaction-specific branding graphic for the Electronic PostMark.


CANADA POST
POSTES CANADA
Electronic Postmark™
Signed by:
John Q. Public
Date: 2005-10-02
Time: 14:02:56
Receipt ID: aF6J4m3BoA
Notice: This signed document has been verified, PostMarked, and logged by the Canada Post EPM Service. The Receipt ID may be used in the future to verify this document's authenticity.

Format – This attribute specifies the desired format of the PostMark image to be returned. Examples of valid values are: JPG, GIF, PNG, BMP, etc … The default is JPG.

Size – This optional attribute instructs the EPM to return one of the supported sizes available from this post's EPM implementation. Valid values are `Small`, `Medium`, and `Large`. Each size is specified by the Postal Administration in pixel width and height and is consistent across Postal EPM implementations. Please consult your local Postal Administration for details. This information can also be extracted from the

Postal “Yellow Pages” directory using the RetrievePostalAttributes operation.

5.4.7 PostMarkedReceipt (XMLDSIG considerations)

The commentary which follows applies only to `PostMarkedReceipt`’s whose `SignatureType` is XMLDSIG and covers the topic of PostMarks over both signatures and data. The `PostMarkedReceipt` is itself an XML Digital Signature ...

An XMLDSIG-formatted `PostMarkedReceipt` is a superset of a conventional RFC 3161 TimeStampToken, and references both a `TstInfo` and a `Receipt` element, as well as a reference to the signature or data being PostMarked, and is represented by a standard detached XMLDSIG `Signature` structure. A PostMark can be over either a signature or just data. When the PostMark targets a signature, the last Reference element of its SignedInfo will point to the SignatureValue element of that target signature. Please refer to Annex B (Informative) Examples for further details.

When the PostMark targets data, the last Reference element of its SignedInfo will point to that detached data, and is, as a consequence, a detached signature. Please refer to Example 2 – Standalone <PostMarkedReceipt> over Data when using PostMark operation. This approach allows posts to use conventional XMLDSIG libraries to create and validate the PostMark signature. The following describes how the child elements of this standard `Signature` representing this PostMarkedReceipt signature are formatted.

KeyInfo – A KeyName child element of KeyInfo identifies the certificate associated with the timestamp's signature and MAY be used to locate, retrieve, and validate the timestamp token's public verification key. This is not necessary if the EPM itself is acting as the TSA and controls the TSA's private and public keys (i.e. trusts itself). It will however always be included for reference. `Reference`'s of `SignedInfo`. There are exactly 3 references in the `PostMarkedReceipt` signature.

1st <Reference> element – will reference the 1st `Object` element containing the `TstInfo` structure as is the case in a conventional RFC 3161 timestamp token. This structure has been adopted directly from the OASIS Digital Signature Services standard and both uses its schema and references its namespace. Please consult this OASIS document for details.

A `TstInfo` element contained in an `Object` element referred to by the first `Reference` will be created by the EPM. The TstInfo element is patterned directly after the TstInfo structure of RFC 3161 and is the same layout as that used in the OASIS DSS standard. EPM implementations shall include this element as a child of the first `Object` element of the signature.

2nd <Reference> element – will reference the 2nd `Object` element containing the `Receipt` structure itself prepared and initialized by the EPM.

Similarly a 2nd `Object` element exist for the `Receipt` structure which includes the `Receipt`, and the `TimeStampToken` contained therein. The `ReceiptSignature` element is not required as the `PostMarkedReceipt` structure is itself an XMLDSIG signature and therefore possess content integrity. This `ReceiptSignature` element is only optional for XMLDSIG-based receipts. The `TstInfo` and `Receipt` structures are described in Examples 1 and 2 in Annex B (Informative) Examples

3rd <Reference> element – is one whose URI attribute references either detached data as would be the case in a direct PostMark operation over some content, or the `SignatureValue`(s) of the specific `Signature`(s) being verified (as in a Verify), or signed (as in a Sign).

On a Verify operation which requested a PostMark, the 3rd `Object` element will contain the `SignatureValue` element(s) of the signature(s) being PostMarked. The EPM will Verify all signatures specified in the `SignatureSelector` element. If the `SignatureSelector` element is omitted, the EPM will attempt to locate and verify all signatures contained in the incoming signed document. Please refer to Example 3 - Embedded <PostMarkedReceipt> over a Verified Signature in Annex B (Informative) Examples.

On a PostMark operation over data content passed in by the user, this 3rd `Object Reference` will contain the data being PostMarked. Please refer to Annex B (Informative) Examples under Example 2 – Standalone <PostMarkedReceipt> over Data when using PostMark operation.

The signature process itself will inherently calculate the `DigestValue` over these 3 `Reference` elements and therefore possesses cryptographic integrity.

*NOTE In order to support embedding of the `PostMarkedReceipt` signature in the incoming XML signature, users shall ensure the incoming signed document will not be invalidated by the inclusion of the `PostMarkedReceipt` signature structure itself, as would be the case with an incoming **enveloped** signature. For this reason users wishing to have their PostMarks `embedded` in their XMLDSIG signed documents should utilize **detached** XML digital signatures in their original documents to ensure that the introduction of the `PostMarkedReceipt` does not invalidate the signature. In this manner the introduction of the PostMark signature will not disturb the integrity of the original signed document being verified. Other approaches can be explored by the client which allow the inclusion of the `PostMarkedReceipt` without invalidating the original signature. This could be accomplished by explicit `Transform`'s which exclude appropriate nodesets.*

5.4.8 OriginalContentType

The complex type `OriginalContent` is used in the CheckIntegrityRequest. The `MimeType` attribute describes the content type of what is being compared against the original content referred to in the referenced `TransactionKey` element. The most common use-case is to pass up the originally signed data (normally the original document or a hash of the original document), to be re-checked. Valid values for the `MimeType` attribute are:

- `text/plain`
- `application/octet-stream`
- `application/vnd.upu-digest-value`
- `application/timestamp-token`
- `application/pkcs7-signature`
- `text/xml`

When either of the first two `MimeType`'s, i.e. **text/plain**, or **application/octet-stream** are specified, the client is expected to pass in the original eContent over which the original signature was created and subsequently verified. This eContent is usually referred to as the SignatureContent or the `SignedInfo` Reference. It is meaningful as a `MimeType` on a previous **Sign, Verify, or PostMark** operation. For XMLDSIG-based checks, the caller should pass in the eContent referred to in each of the XMLDSIG `Reference` elements that are present in the signature being checked.

When a `MimeType` of **application/vnd.upu-digest-value** is specified, the actual value of the hash calculation over the originally signed content shall be passed in and compared.

When a `MimeType` of **application/timestamp-token** is specified, the client is expected to pass in the detached binary RFC 3161 `TimeStampToken` element contained in the `PostMarkedReceipt` structure. This is normally inside the `PostMarkedReceipt` returned on a Verify operation which requested a receipt via the `IssuePostMarkedReceipt` option. It is optionally present in XMLDSIG-based signatures.

When a `MimeType` of **application/pkcs7-signature** is passed, the entire PKCS7 signature shall be passed in and will be compared.

When a `MimeType` of **text/xml** representing an XMLDSIG signature is passed in, the contents of the `dsig:SignatureValue` element, excluding the bounding tags, shall be passed in and will be compared.

```
<xs:complexType name="OriginalContentType">
   <xs:simpleContent>
      <xs:extension base="xs:base64Binary">
         <xs:attribute name="MimeType" type="xs:string"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
```

5.4.9 ParticipatingPartyType

When working within an EPM lifecycle, the EPM can restrict the access of all transaction information to a finite set of parties termed the Participating Parties. This complex type represents the group of eligible individuals or organizations who are permitted access or allowed to contribute to this lifecycle. It can be specified when either

1) restriction of a lifecycle of business events to a selected set of parties is required,

2) Delegated Confidentiality is being used, or

3) Proof-of-Delivery (POD), Proof-of-Possession (POP) is required by the stakeholders

NOTE ClaimedIdentity used on its own can also provide POD/POP support.

Each `ParticipatingParty` would be accessing the EPM Service utilizing the specified `AccessLevel`. This element is also related to the `AccessScope` element which defines the overall scope of the permitted access and at which level access should be granted. Each entry, if specified, shall be initialized with the DistinguishedName of the party who is permitted to either access or contribute to the Lifecycle.

EXAMPLE `CN=Joe Public,O=Acme,OU=Purchasing,C=CA`.

Each `ParticipatingParty` identified by `PartyName` would be accessing the EPMService utilizing the specified `AccessLevel`. Valid string values for `AccessLevel` are `Default` and `Signed`. When `Default` is specified, the `PartyName` can access this LifeCycle after having authenticated over the primary authentication mechanism for this postal implementation. For example, in Canada that would be HTTP Basic Authentication as supported in the Web server. When `Signed` is specified, the `PartyName` shall initialize the `RequesterSignature` element of `ClaimedIdentity` in order to execute operations against this Lifecycle. In either case the authenticated user string value shall match the PartyName string value specified in this StartLifecycle request (see below for examples). Similarly when the `AccessScope` element is marked as `Organizational` then the authenticated user's organization (derived from either the certificate or information supplied at registration time) shall match the `PartyName` value specified this `PartyName` entry. Any party from that organization, once authenticated, can participate in this Lifecycle.

EXAMPLE sample values:

1) for AccessLevel = Signed and AccessScope = Individual, PartyName values might look like this: CN=Joe Public,O=Acme,OU=Purchasing,C=CA

2) for AccessLevel = Signed and AccessScope = Organizational, PartyName values might look like this: O=Acme,OU=Purchasing,C=CA

3) for AccessLevel = Default and AccessScope = Individual, PartyName values might look like this: Joe Public or CN=Joe Public

4) for AccessLevel = Default and AccessScope = Organizational, PartyName values might look like this: O=Acme,OU=Purchasing,C=CA

NOTE For `AccessScope = Organizational` and `AccessLevel = Default`, implementations would derive the authenticated user's organization from information captured at registration time, normally kept in the EPM's registration database. The design of this registration sub-system is beyond the scope of the EPM Specification.

The `NotifyEvents` element holds a list of the operations within this Lifecycle for which this Party will be notified e.g. Verify, CheckIntegrity, and LogEvent. The `ContactID` element contains the eMail address of the ParticipatingParty and will be used to contact the `PartyName` whenever any of the operations in the `NotifyEvents` list occurs.

```
   <xs:complexType name="ParticipatingPartyType">
      <xs:sequence>
         <xs:element name="PartyName" type=" epm:PartyNameType"/>
         <xs:element name="AccessLevel" type="xs:string"/>
         <xs:element name="NotifyEvents" type="epm:ValidOperation" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="ContactID" type="xs:string" nillable="true"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="PartyNameType">
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="ScopeQualifier" type="xs:string" use="optional"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
```

5.4.10 ClaimedIdentity

ClaimedIdentity is an optional input element within the EPM and serves two main purposes.

1) as a verb-specific Proof-of-Delivery / Proof-of-Possession mechanism required by originators or recipients.

2) as an alternate authentication mechanism in support of higher strength non-repudiability when `AccessLevel` on the `ParticipatingParty` element of `StartLifecycle` is set to `Signed`.

In the Proof-of-Delivery scenarios, the client is signing over content with their private signing key, as part of, for example, a CheckIntegrity operation. The content over which they are signing, in this example, is the hash of the OriginalContent element of the CheckIntegrity operation. As such they are irrefutably attesting to having possessed the content (document) that they have presumably received from the signer at origin. Please refer to subclause 5.5.4 entitled CheckIntegrity for additional details on this scenario.

```
   <xs:complexType name="ClaimedIdentityType">
      <xs:sequence>
         <xs:element name="Name" type="epm:NameIdentifierType"/>
         <xs:element name="SupportingInfo" type="epm:SupportingInfoType"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="NameIdentifierType">
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="NameQualifier" type="xs:string" use="optional"/>
            <xs:attribute name="Format" type="xs:anyURI" use="optional"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

   <xs:complexType name="SupportingInfoType">
      <xs:sequence>
         <xs:element name="BasicAuth" type="epm:BasicAuthType" nillable="true"/>
         <xs:element name="RequesterSignature" type="epm:QualifiedDataType"
nillable="true"/>
         <xs:element name="AlternateIdentity" type="epm:AlternateIdentityType"
nillable="true"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="BasicAuthType">
      <xs:sequence>
         <xs:element name="UserID" type="xs:string"/>
         <xs:element name="Password" type="xs:string" nillable="true"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="AlternateIdentityType" abstract="true">
      <xs:sequence>
         <xs:element name="IdentityToken" type="xs:anyType"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="YourFavoriteIdentityTokenType">
      <xs:complexContent>
         <xs:extension base="epm:AlternateIdentityType">
            <xs:sequence>
               <xs:element name="FirstElement" type="xs:string"/>
               <xs:element name="SecondElement" type="xs:base64Binary"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

The NameIdentifierType above is taken directly from SAML and included in this schema for convenience. The NameIdentifierType is used where different types of names are needed (such as email addresses, Distinguished Names, etc). This type is borrowed from SAMLCore1.1 (http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf). It consists of a string with the attributes as indicated below.

NameQualifier – The security or administrative domain that qualifies the name of the subject. This attribute provides a means to federate names from disparate user stores without collision.

Format – A URI reference representing the format in which the string is provided.

AlternateIdentityType – This abstract type (i.e. `abstract="true"`) cannot be directly implemented. One needs to extended from this base type as shown in the YourFavoriteIdentityTokenType example above.

See subclause 7.3 of SAMLCore1.1 (http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf) for URI references that may be used as the value of the Format attribute. Example: urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName (as defined in XMLDSIG) or urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress (as defined in addr-spec of RFC 2822).

RequesterSignature – Under circumstances when RequesterSignature shall be initialized, as in a Lifecycle whose `AccessLevel` has been specified as `Signed`, if the `SignatureType` is specified as XMLDSIG, the `RequesterSignature` should be created as an enveloped XMLDSIG signature over both the `TransactionKey` and the `OrganizationID` elements. The resulting signature shall then be included in the `RequesterSignature` element of the Request.

The example below is when higher non-repudiability is requested by the owner of the Lifecycle. In this scenario each `ParticipatingParty` is designated, by the owner, to have access to, and be allowed to contribute to, and participate in, this Lifecycle. The owner is able to set the `AccessLevel` for this Lifecycle to `Signed`, which means that each ParticipatingParty, presumably adding events to the

Lifecycle, shall create a `RequesterSignature` within the `ClaimedIdentity` element. This signature can be either a PKCS7 enveloping signature or an XMLDSIG enveloped signature over the `TransactionKey` and `OrganizationID` elements. Please refer to Example 4 - RequesterSignature over TransactionKey for any operation in protected Lifecycle in the Examples subclause.

RequesterSignature for SignatureType PKCS7

When the RequesterSignature is of SignatureType PKCS7, implementations shall follow the serialization rules below. When signing XML elements which may have been serialized from a SOAP envelope, care shall be taken to ensure that the content stream is identical across toolkits. For this reason, the TransactionKey and OrganizationID formatting rules shall be as follows:

Serialization Conventions

- only leaf node elements will have their start tag, end tag, and content all on the same line
- elements with child nodes will have their start tag and end tag on a line by themselves and will wrap their descendants
- no pretty formatting or indenting will exist on any line
- carriage return line feed shall be used as the 2 character line separator (i.e. x'0d' x'0a')
- the first and last character of the eContent to the signature will be the opening < and the closing > respectively
- all attributes (with the exception of `MimeType`) will be dropped from tags (e.g. xsi:type="xsd:string", etc ...)

EXAMPLE for `SignatureType` PKCS7

```
<TransactionKey>
<Locator>
<CountryCode>CA</CountryCode>
<Version>114</Version>
<ServiceProvider></ServiceProvider>
<Environment></Environment>
</Locator>
<Key>0311352e0C3</Key>
<Sequence>1</Sequence>
</TransactionKey>
<OrganizationID>Acme Corporation</OrganizationID>
```

The resultant PKCS7 signature shall contain the above eContent (i.e. is not a detached signature).

NOTE This is not an issue with XMLDSIG based signatures since the canonicalization which takes place normalizes subtle formatting, tabbing, line wrapping, and white space differences.

5.4.11 AccessScope and Scopes

This element, which is specified in the StartLifecycle operation, determines the scope of who is allowed to access the contents of, or contribute to, this Lifecycle. Valid values are `Global`, `Organizational`, `Individual, or Mixed`. `Global` grants access to anyone. `Global` allows any user to access the Lifecycle. `Organizational` grants access to anyone within the Organization, as initialized in the `ParticipatingParty` element. For example, when a post is using strong authentication, access privilege can be determined by comparing the DistinguisedName field of the certificate used to sign the request against any of the occurrences of `ParticipatingParty`. `Individual` grants access to only the selected `Individual`'s, again in or example determined by the DistinguisedName field of the certificate used to sign the request. See also subclause 5.4.9 entitled ParticipatingPartyType for further details. The party who issues the request shall authenticate that request if the AccessScope element is **not** marked as `Global`. That is, if `AccessScope` is marked as `Individual`, there shall exist a corresponding `ParticipatingParty` element, and the individual making the request shall be in that list. Likewise, if the `AccessScope` is marked as `Organizational`, then the individual authenticating the request shall be from an organization in the `ParticipatingParty` list. This identity is either compared against the DistinguisedName field of the certificate used to sign the request, or the pre-registered user that authenticated with the EPM. `Mixed` allows for both `Organizational` and `Individual` entries in the `ParticipatingParty` list.

```
<xs:simpleType name="Scopes">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Global"/>
        <xs:enumeration value="Organizational"/>
        <xs:enumeration value="Individual"/>
        <xs:enumeration value="Mixed"/>
    </xs:restriction>
</xs:simpleType>
```

This simple object type `Scopes>` represents a list of access levels that is used to restrict groups or individuals who are allowed to contribute to, or access the contents of events within a given lifecycle. These scope values are used by the EPM Service to specify the granularity of the access rights for this lifecycle and its contents. This simple object is used by the participating parties construct (Please refer to ParticipatingPartyType in subclause 5.4.9 for details) and is references through its `AccessScopes` element. The following are the permitted values:

Global – A string element representing 'Global' access to the events within this lifecycle. Global grants access of the transactions within a lifecycle to anyone. 'Global' requests need not be signed.

Organizational – A string element representing 'Organizational' access to the events within this lifecycle. Organizational grants access to anyone within the Organization to the events within this lifecycle. Privilege is determined by comparing the DistinguishedName field of the certificate used to sign the request against any of the occurrences of ParticipatingParty element.

Individual – A string element representing 'Individual' access to the events within this lifecycle. Individual grants access to only the selected 'Individual's, again determined by the DistinguishedName field of the certificate used to sign the request.

Mixed – A string element representing 'Mixed' access to the events within this lifecycle. This mode states that the `PartyName`'s may be any of the above types. The specific type is specified on the `PartyName` element.

5.4.12 EncryptResponse Option

This option is utilized on three possible EPM operations, a Verify, a RetrieveResults, and a Decrypt, and is therefore covered here for reference.

EncryptResponse – This is a Special Case optional usage of the EPMs confidentiality capability, and works in conjunction with a local encrypt operation issued by an individual client customer using the EPM's public encryption key. It is termed "Delegated Confidentiality" within the EPM context. This capability frees individuals from having to manage the public keys of intended recipients. When a subscriber wishes to encrypt content for confidentiality reasons, they simply encrypt that content locally with a public key provided to them by the Postal Administration. This single public key belongs to the post and is the only public key the customer needs to maintain on their desktop or within their application. After having encrypted the content with this post-specific public key, the content is now secured for transport. This envelope can be sent to the recipient.

At the recipients end, the recipient does not have the private key required to decrypt this envelope they just received. Consequently they needs to ask the post's EPM Service to Decrypt it for them. It would not make sense for the EPM to simply Decrypt the content and pass it back to the caller in the clear. To prevent this exposure, the EPM "encrypts the response" when the `EncryptResponse` option is turned on in the recipient's request. The EPM however requires the callers public key in order to be able to actually encrypt the response prior to returning it (who is the recipient in this scenario). In order to provide the EPM with the callers public key, the caller signs the request.

There exist scenarios whereby the EncryptResponse is utilized on a Verify request, a RetrieveResults request, or a Decrypt request. can be exercised where the recipient turns on the `EncryptResponse` option on a RetrieveResults operation and picks up a document signed by an originator which was left with the EPM. This usage is termed "Sign for Pickup" which may also optionally employ the `EncryptResponse` option.

Yet another scenario involves a CheckIntegrity call by the recipient whereby the content is passed to the EPM for comparison against its non-repudiation store to ascertain authenticity. Since the recipient is in possession of the content when the call is made, Proof-of-Delivery and Proof-of-Possession are also supported.

As such there are three scenarios involving end-to-end confidentiality which could be considered. They differ primarily in how the document is transported and which operation the recipient uses. Only the first ensures irrefutable Proof-of-Delivery and Proof-of-Possession.

1. **Scenario:** "Sender encrypts document with EPM public key and delivers document directly to recipient". Sequence of events as follows:
    - Sender locally signs document
    - Sender locally encrypts signed document with the EPM's public key
    - Sender calls EPM to Verify the document using the following options
        - IssuePostMarkedReceipt
        - DecryptIncomingEnvelope
        - EncryptResponse to protect the updated signature
    - Sender decrypts Verify response using their own private key and checks status
    - Sender re-encrypts signed document for recipient using recipient's public key (assumed to be either available or may be retrieved using the EPM's Locate operation)
    - Sender mails signed and encrypted document to recipient
    - Recipient decrypts document with their own private key
    - Recipient locally encrypts signed and PostMarked document with the EPM's public key
    - Recipient issues signed CheckIntegrity request with the following option
        - DecryptIncomingEnvelope
    - Recipient checks status

2. **Scenario:** "Sender encrypts document with EPM public key and deposits document with EPM for pickup". Sequence of events as follows:
    - Sender locally signs document
    - Sender locally encrypts signed document with the EPM's public key
    - Sender calls EPM to Verify the document using the following options
        - IssuePostMarkedReceipt
        - DecryptIncomingEnvelope
        - EncryptResponse to protect the updated signature
    - Sender checks status
    - Sender notifies recipient that they can pickup the signed document using this specific

TransactionKey which they pass to them "out of band"

- Recipient issues signed RetrieveResults request with the EncryptResponse option
- Recipient decrypts response using their own private key to extract the document

3. **Scenario:** "Sender Verifies document, re-encrypts verified document with recipient's public key, and delivers document directly to recipient". Sequence of events as follows:
   - Sender locally signs document
   - Sender locally encrypts signed document with EPM public key
   - Sender calls EPM to Verify the document using the following options
     - IssuePostMarkedReceipt
     - DecryptIncomingEnvelope
     - EncryptResponse
   - Sender decrypts Verify response using their own private key and checks status
   - Sender re-encrypts signed document for recipient using recipient's public key (assumed to be either available or may be retrieved using the EPM's Locate operation)
   - Sender mails signed and encrypted document to recipient
   - Recipient decrypts document with their own private key
   - Recipient locally verifies signature and optionally the PostMark

NOTE In situations where the EPM is deployed centrally as a shared service, the 2nd scenario above should be used with the `ParticipatingParty` *feature of StartLifecycle to ensure that only intended recipients are receiving confidential content. Additionally EPM implementations SHALL ensure that the authenticated requests which involve decryption of sensitive content honor and respect restricted* `ParticipatingParty` *lists within that Lifecycle.*

5.4.13 ContentMetadata

This element can be optionally passed as input on most operations and can be used by the client implementations to provide further contextual information as desired. Possible uses include: file name, file date, file size, file owner information, special attributes pertaining to the SignedInfo, etc ... It is also used within the `Receipt` element of the `PostMarkedReceipt` structure to allow individual Posts to extend the information contained in a `Receipt`.

```
<xs:complexType name="ContentMetadataType">
   <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Value" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

5.4.14 ValidOperation

The 5 core operations below shall be supported by the implementation in order to be considered an UPU-endorsed EPM. Support for lifecycle can be provided implicitly through the TransactionKey in order to gain EPM endorsement. Explicit support of the StartLifecycle operation is not strictly required. This element is also referenced by the NotifyEvents type. ValidOperation is used in NotifyEvents.

```
<xs:simpleType name="ValidOperation">
   <xs:restriction base="xs:string">
      <xs:enumeration value="Verify"/>
```

```
          <xs:enumeration value="PostMark"/>
          <xs:enumeration value="CheckIntegrity"/>
          <xs:enumeration value="RetrieveResults"/>
          <xs:enumeration value="Sign"/>
<!-- The 7 optional operations below are considered extensions
and may optionally be implemented by local Postal Admins. -->
          <xs:enumeration value="StartLifecycle"/>
          <xs:enumeration value="LogEvent"/>
          <xs:enumeration value="Encrypt"/>
          <xs:enumeration value="Decrypt"/>
          <xs:enumeration value="Locate"/>
          <xs:enumeration value="RetrieveSummary"/>
          <xs:enumeration value="RetrievePostalAttributes"/>
       </xs:restriction>
   </xs:simpleType>
```

5.4.15 ValidOption

ValidOption is used in OperationOptions, and constrains the list of valid options.

```
   <xs:simpleType name="ValidOption">
       <xs:restriction base="xs:string">
          <xs:enumeration value="EndLifecycle"/>
          <xs:enumeration value="ExtendLifecycle"/>
          <xs:enumeration value="VerifyCertificate"/>
          <xs:enumeration value="DecryptIncomingEnvelope"/>
          <xs:enumeration value="EncryptResponse"/>
          <xs:enumeration value="StoreNonRepudiationEvidence"/>
          <xs:enumeration value="IssuePostMarkedReceipt"/>
          <xs:enumeration value="ReturnTimeStampAudit"/>
          <xs:enumeration value="ReturnSignatureInfo"/>
          <xs:enumeration value="ReturnX509Info"/>
       </xs:restriction>
   </xs:simpleType>
```

5.4.16 Event

This simple object type represents a list of all valid operations that the EPM Service will honor in support of the NotifyEvents element within the StartLifecycle operation (See Lifecycle Management in subclause 5.3.3 for more information.). This simple object is used by the participating parties (refer to ParticipatingPartyType in subclause 5.4.9 for details) and is referenced through it's 'NotifyEvents' element. The following are the valid values that this element supports:

- Verify
- PostMark
- CheckIntegrity
- RetrieveResults
- Sign
- Encrypt
- Decrypt
- Locate
- LogEvent
- StartLifecycle
- RetrieveSummary

- RetrievePostalAttributes

5.4.17 ClientApplication

```
The ClientApplicationType is used in most requests to indicate the desktop client and
version from which the EPM operation originated. Examples include: MS Word Extension
V2.23, and the Acrobat Plugin V7.18. This element (through its attribute) also
identifies the scheme used to transform the document. That is, process any inclusion
and exclusion of document parts, as well as any optional transformation of that
content which may have been applied and subsequently used as input to the hash
algorithm. If this element is omitted, it is assumed that the entire document was used
as input to the hash calculation. It's use is optional, and when not specified, the
entire document will be assumed. <xs:complexType name="ClientApplicationType">
      <xs:sequence>
         <xs:element name="NameAndVersion" type="xs:string"/>
         <xs:element name="ContentTransformScheme"
type="epm:ContentTransformSchemeType" nillable="true"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="ContentTransformSchemeType">
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="ContentTransformSchemeURI" type="xs:anyURI"
use="optional"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
```

5.4.18 ContentIdentifier

This optional element can be used to provide an application-specific identifier for the type of form, document, or file being operated on. Organizational level retrieval access and privileges can be set based on this element when using the RetrieveSummary operation. This string element might contain values such as `PDF`, `Engineering Drawings`, `Form 628`, etc … or anything the client may wish to filter transactions on.

5.4.19 Version

This mandatory element shall be present on every EPM request (except the RetrievePostalAttributes which already contains the mandatory `Locator` element which itself contains the `Version` element) and is used to help identify the version of the EPM Interface Specification schema this request is formatted as. It is a string element and for example might contain: 1.14, 1.15, 2.0, etc … Posts may use this element in conjunctions with the URL of the EPM Service itself to identify the version of the EPM Interface that this call is formatted under.

5.4.20 SignaturePolicyIdentifier

This abstract type cannot be directly implemented. One needs to extended from this base type as shown in the SomeSignaturePolicyIdentifierType example below. This example is a simplified version of the SignaturePolicyIdentifierType in ETSI's XAdES schema TS 101 903. Possible use of this abstract type might be in an implementation where the local jurisdiction mandates the explicit use of disclosed signature policies. The SignaturePolicyIdentifier could be returned as an additional signed reference or property in, for example, a Sign response. An XMLDSIG Reference to this type could be specified as part of a signing template passed in on a Sign operation.

```
   <xs:complexType name="SignaturePolicyIdentifierType" abstract="true">
```

```
       <xs:sequence>
          <xs:element name="SignaturePolicyIdentifier" type="xs:anyType"/>
       </xs:sequence>
    </xs:complexType>
    <xs:complexType name="SomeSignaturePolicyIdentifierType">
       <xs:complexContent>
          <xs:extension base="epm:SignaturePolicyIdentifierType">
             <xs:sequence>
                <xs:element name="SigPolicyID" type="xs:anyURI"/>
                <xs:element name="SigPolicyURL" type="xs:string"/>
                <xs:element name="SigPolicyHashAlgo" type="xs:anyURI"/>
                <xs:element name="SigPolicyHashValue" type="xs:string"/>
                <xs:element name="SigPolicyUserNotice" type="xs:string"
nillable="true"/>
             </xs:sequence>
          </xs:extension>
       </xs:complexContent>
    </xs:complexType>
```

5.5 EPM Interface Specification – "Core" Operations in Detail

The following operations have been identified as required support services to be qualified as complaint to the UPU EPM Core Specification:

- Verify
- CheckIntegrity
- PostMark
- RetrieveResults
- Sign

A description of each of these operations and how to process them in conjunction with an EPM Web-based Service follows.

5.5.1 Verify

5.5.1.1 Verify Edit Rules Summary

The Verify operation is normally invoked by the originator of the document, although it can be invoked by the recipient as well. It is also normally the first event in the Lifecycle (after the StartLifecycle if specified). After the document is Verified and optionally PostMarked at origin, it then can be checked using the Verify or CheckIntegrity operation.

The Verify operation requires, as input, a PKCS7 or XMLDSIG signature object and an optional PostMarkedReceipt. The EPM Service will perform a Verification on the signature object and optionally on the PostMarkedReceipt's signature(s). A Verify will perform an OCSP or CRL check to validate the signing certificate when the VerifyCertificate option is specified.

5.5.1.2 VerifyOptions Request Flags

The following option flags can be set in the VerifyOptionsType element. These elements are normally of type boolean (the `IssuePostMarkedReceipt` being an exception) and should be set to 'True' or 'False', the default value being 'False'. This object will be added as an element reference to the `VerifyRequest` element which is referenced in the next subclause.

```
   <xs:complexType name="VerifyOptionsType">
      <xs:sequence>
         <xs:element name="EndLifecycle" type="xs:boolean"/>
         <xs:element name="ExtendLifecycle" type="xs:boolean"/>
         <xs:element name="VerifyCertificate" type="xs:boolean"/>
         <xs:element name="DecryptIncomingEnvelope" type="xs:boolean"/>
         <xs:element name="EncryptResponse" type="xs:boolean"/>
         <xs:element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
         <xs:element name="IssuePostMarkedReceipt"
type="epm:IssuePostMarkedReceiptType" nillable="true"/>
         <xs:element name="ReturnSignatureInfo" type="xs:boolean"/>
         <xs:element name="ReturnX509Info" type="xs:boolean"/>
      </xs:sequence>
   </xs:complexType>
```

EndLifecycle – Set to 'True' to end the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

VerifyCertificate – Set to 'True' to enable certificate status checking. If set to 'False', the certificate status checking will be bypassed, but the minimal certificate validation will still occur (e.g. from/to expiry check, certificate present check, etc …). All certificate validation results will be returned in the X509InfoType complex element. Please refer to subclause 5.4.5 entitled X509InfoType for details..

DecryptIncomingEnvelope – Set to 'True' to instruct the EPM Service to decrypt the following incoming elements before performing its Verification: `SignedConent` and `SignatureData`. These 2 elements included as EnvelopedData objects shall be encrypted with the public key portion of the EPM Service's private decryption key. This option is used to ensure confidential delivery of the transaction content to the EPM Service. See also "EncryptResponse" below.

EncryptResponse – This option requires that the caller sign the request by initializing the `ClaimedIdentity` request element. This option instructs the EPM to encrypt the `SignatureData` element before returning it using the public key present in the incoming `RequesterSignature`. Additionally if the caller has requested that the `SignatureInfo` be returned, then the `SignedContent` sub-element should also be encrypted. Please also refer to subclause 5.4.12 entitled EncryptResponse Option which covers the use of this option with examples of usage as they apply to the Verify, Decrypt, and RetrieveResults operations.

StoreNonRepudiationEvidence – Set to 'True' to turn on database logging of non-repudiation evidence. If set to 'False', non-repudiation information is not logged to the database. If set to 'False' a TransactionKey is still generated but only skeleton information relating to the transaction status is logged to the database. This skeleton information is logged solely in order to support subsequent retrieval operations and to allow this event to participate in Lifecycles.

IssuePostMarkedReceipt – If included as part of the Verify request, will cause the EPM to return a receipt attesting to the validity and non-repudiability of the Verify operation. This returned receipt is PostMarked using an EPM-generated timestamp. Please refer to subclause 5.4.6 entitled PostMarkedReceipt for details. This timestamp and other receipt information is returned in a `PostMarkedReceipt` element which is itself bound with a signature of authenticity. Valid values for the `Location` sub-element are `standalone` and `embedded` and instructs the EPM where to place the resulting `PostMarkedReceipt` element. For PKCS7 and XMLDSIG based signatures, a value of `standalone` instructs the EPM to return a separate standalone `PostMarkedReceipt` as shown in Example 1 – Standalone <PostMarkedReceipt> over a Verified Signature. For XMLDSIG signatures, a value of `embedded` instructs the EPM to embed a `PostMarkedReceipt` structure into the incoming signature upon successful verification as shown in Example 3 - Embedded <PostMarkedReceipt> over a Verified Signature. This updated signed document, which is now also PostMarked, is returned in the `SignatureData` element of the verify response. Lastly for PKCS7 signatures, a value of `embedded` instructs the EPM to also embed an RFC 3161 binary TimeStampToken into the incoming signature upon successful verification. The verified and updated PKCS7 signature now containing the embedded RFC 3161 timestamptoken is returned in the `SignatureData` element of the response, and the `PostMarkedReceipt` is returned as requested. Essentially the `embedded`, as applied to the verification of PKCS7 signatures really means *"Create and return a PostMarkedReceipt structure, and also embed an RFC 3161 timestamp token into my signature".* This overloaded meaning is necessary when verifying XMLDSIG based signatures since the `PostMarkedReceipt` can much more naturally be `embedded` into the incoming signed document.

```
<xs:complexType> name="IssuePostMarkedReceiptType">
   <xs:sequence>
      <xs:element name="Location" type="xs:string"/>
      <xs:element name="PostMarkImageSize" type="epm:ValidImageSize"/>
      <xs:element name="PostMarkImageType" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

NOTE   If an embedded timestamptoken already exists within an incoming PKCS7 signature, the EPM implementation should verify it but shall not replace it. As described above, the separately returned `PostMarkedReceipt` serves as the receipt and timestamp for all subsequent Verify operations and any number can be saved by the calling application.

ReturnSignatureInfo – Set to 'True' to return a detailed breakdown of the signature object, including the original content. Please refer to subclause 5.4.4 entitled SignatureInfoType for details.

ReturnX509Info – Set to 'True' to return a detailed breakdown of the certificate used to perform the operation. Please refer to subclause 5.4.5 entitled X509InfoType for details.

5.5.1.3 Verify Request Elements

Set the following parameters or elements of the VerifyRequestType element before invoking the EPM Service as appropriate.

```
   <xs:element name="VerifyRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="VerifyOptions" type="epm:VerifyOptionsType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"
nillable="true"/>
            <xs:element name="ClaimedIdentity" type="epm:ClaimedIdentityType"
nillable="true"/>
            <xs:element name="OrganizationID" type="xs:string" nillable="true" />
            <xs:element name="ClientApplication" type="epm:ClientApplicationType"/>
            <xs:element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <xs:element name="SignatureData" type="epm:QualifiedDataType"
nillable="true"/>
            <xs:element name="SignedContent" type="epm:QualifiedDataType"
nillable="true"/>
            <xs:element name="PostMarkedReceipt" type="epm:PostMarkedReceiptType"
nillable="true"/>
            <xs:element name="SignatureSelector" type="epm:SignatureSelectorType"
nillable="true"/>
            <xs:element name="ContentMetadata" type="epm:ContentMetadataType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

VerifyOptions – An element whose type is a complex element defined by `VerifyOptionsType` from the previous subclause.

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. `Locator`, `Key`, and `Sequence` elements should be initialized as 'null' for Verify requests unless one is extending a Lifecycle. Please refer to TransactionKeyType in subclause 5.4.2.

ClaimedIdentity – The RequesterSignature element of ClaimedIdentity shall be initialized when either Proof-of-Delivery or Proof-of-Possession is desired by the transacting parties, or when this operation is part of a `StartLifecycle` whose `AccessLevel` is `Signed.` See also

ClaimedIdentity in subclause 5.4.10.

OrganizationID – A string element representing the identifier of the organization requesting the service. This value shall match against a list of valid Organization Identifiers that are registered with the EPM Service.

ClientApplication – See ClientApplication subclause 5.4.17

ContentIdentifier – See ContentIdentifier subclause 5.4.18

SignatureData – The CMS-compliant PKCS7 to be verified in either encapsulated ASN.1 binary PKCS7v1.5 or the XML Digital Signature Syntax and Processing standard, RFC 3275. In either case, the application shall present the signature to be verified as an octet-stream to the SOAP layer which will base64 encode it for transport. The EPM will determine the signature format from the `MimeType` attribute. Valid values are either `text/xml`, in which case the EPM will assume that the caller is passing in an XMLDSIG-formatted signature, `application/pkcs7-signature`, in which case the EPM will assume the caller is passing in ASN.1 binary PKCS7, or `application/pdf` in which case the EPM will assume a signed PDF.

MimeType – This is an attribute of the `SignatureData` element above and shall specify either `text/xml` if Data contains an XMLDSIG signature, or `application/pkcs7-signature` if `SignatureData` contains a PKCS7 SignedData object.

PostMarkedReceipt – This optional element allows the client caller to optionally pass up both the signature to be verified (in the `SignatureData` element) and the `PostMarkedReceipt`. In this manner, an EPM implementation can validate both the signature and the `PostMarkedReceipt` over that signature. The `PostMarkedReceipt` passed in should be from the `PostMarkedReceipt` issued as part of the original Verify of this signature. They are cryptographically bound to each other and this binding will be verified by the EPM.

There are 2 scenarios to be addressed by the EPM implementation when Verifying a PostMarkedReceipt. They are described below:

- When Verifying a `PostMarkedReceipt` which was originally created over a signature, the following verification steps shall be performed:
    - Verify the signature contained in the `SignatureData` request element
    - Verify the `ReceiptSignature` element within the incoming `PostMarkedReceipt.` Please note that for PKCS7-based receipts, the `ReceiptSignature` itself is a detached signature over the serialized contents of the `Receipt` element.
    - Lastly an equality check shall be performed to confirm that this `PostMarkedReceipt` is bound to the signature in the incoming `SignatureData.` This is accomplished by extracting the messageImprint (i.e. the 3rd field in the TstInfo structure) from the `TimeStampToken` element of the `PostMarkedReceipt` and comparing it against the hash of the SignatureValue (PKCS1) of the signature in the `SignatureData` element.
- When Verifying a PostMarkedReceipt which was originally created over a data, as would be the case when a PostMark operation was used, the following verification steps shall be performed:
    - Verify the `ReceiptSignature` element within the incoming `PostMarkedReceipt.` Please note that for PKCS7-based receipts, the ReceiptSignature itself is a detached

signature over the serialized contents of the `Receipt` element.

- Perform an equality check to confirm that this `PostMarkedReceipt` is bound to the data in the incoming `SignedContent.` This is accomplished by extracting the messageImprint (i.e. the 3rd field in the TstInfo structure) from the `TimeStampToken` element of the `PostMarkedReceipt` and comparing it against the hash of the data in the `SignedContent` element.

SignedContent –This optional element is required for the verification of detached signatures. If this element contains something (i.e. is not nil), the EPM will assume the signature to be verified is detached. Applies only to PKCS7 based signatures. For XMLDSIG based signatures, the EPM supports same-document detached signatures and expects the data to be part of the signed document.

NOTE When verifying a PostMarkedReceipt, as described in the second scenario above, this element shall contain the data that was originally PostMarked. It is required to confirm the binding of the `TimeStampToken`.

SignatureSelector – The optional `SignatureSelector` element qualifies the XMLDSIG signature(s) to be verified by the EPM. This element may also serve useful if the user in unsure of exactly what has been verified, and wishes to control the verification process more explicitly.

If the user wishes to Verify a particular signature or signatures, they have two choices has to how they may specify the `dsig:Signature` nodes to be verified. Each choice is a sub-element of the `SignatureSelectorType` below.

The **First** method allows users to specify any ancestor (parent) node of the signature(s) to be verified and are specified by including these names as `NodeName` element(s). The value is expressed as a string. A namespace URI qualifier may precede the actual signature `NodeName` value.

```
   <xs:complexType name="SignatureSelectorType">
      <xs:sequence>
         <xs:choice>
            <xs:element name="NodeName" type="xs:string" minOccurs="1"
maxOccurs="unbounded"/>
            <xs:element name="XPathSelector" type="epm:XPathSelectorType"/>
         </xs:choice>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="XPathSelectorType">
      <xs:sequence>
         <xs:element name="XPath" type="xs:string"/>
         <xs:element name="NameSpace" type="xs:string" nillable="true" />
         <xs:element name="Qualifer" type="xs:string" nillable="true"/>
      </xs:sequence>
   </xs:complexType>
```

EXAMPLE The user would specify string values of `lgl:Party1` and/or `lgl:Party2` to explicitly instruct the EPM what to Verify. By default the EPM will search for signature nodes specified as `<dsig:Signature>`, which appear as descendants of the document root.

```
   <lgl:Party1>
      <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      ...
      </dsig:Signature>
   </lgl:Party1>
   <lgl:Party2>
      <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      ...
      </dsig:Signature>
   </lgl:Party2>
```

The **Second** method involves specifying an XPath expression which when evaluated will return the target `<dsig:Signature>` nodes to be verified. The actual Xpath expression is included in the `XPath` element and any required namespace and qualifier can be specified in the `NameSpace` and `Qualifier` elements.

EXAMPLE Using an XPath expression to select the target `<dsig:Signature>` *nodes*

```
   <lgl:Document xmlns:lgl="http://www.lgl.org/SomeService"
                 xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <lgl:Signatures>
         <dsig:Signature>1st</dsig:Signature>
         ...
         <dsig:Signature>2nd</dsig:Signature>
         ...
         <dsig:Signature>3rd</dsig:Signature>
         ...
      </lgl:Signatures>
   </lgl:Document>
```

In the example above a value of `//lgl:Signatures//dsig:Signature[position=2]` would select only the second signature to be verified.

A value of `//lgl:Signatures//dsig:Signature` in the `XPath` element would cause all signatures to be verified.

In both examples a value of `http://www.lgl.org/SomeService` and `lgl` shall be specified for the `NameSpace` and `Qualifier` elements respectively in order to allow the XPath string expression to evaluate.

ContentMetaData – A string element containing custom details of the signed data that can be specified by the client. Example usage could be the original file name, file date, file size or file owner information.

5.5.1.4 Verify Response Object

A VerifyResponseType complex element is populated and returned by the EPM Service. Here are the elements it contains:

```
   <xs:element name="VerifyResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail" 
type="epm:TransactionStatusDetailType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
            <xs:element name="PostMarkedReceipt" type="epm:PostMarkedReceiptType" 
nillable="true"/>
            <xs:element name="SignatureData" type="epm:QualifiedDataType" 
nillable="true"/>
            <xs:element name="SignatureInfo" type="epm:SignatureInfoType" 
nillable="true"/>
            <xs:element name="X509Info" type="epm:X509InfoType" nillable="true"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

TransactionStatus – A string element representing the result of the overall transaction. ‘0’ denotes success. ‘1’ denotes that a warning was generated. Transaction failure is denoted by a specific numerical value..

TransactionStatusDetail – An element whose type is a complex element defined by TransactionStatusDetailType. See TransactionStatus and TransactionStatusDetailType in subclause 5.4.1)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType Please refer to TransactionKeyType in subclause 5.4.2. The Locator, Key, and Sequence elements will be populated and returned by the EPM Service.

PostMarkedReceipt – An optional `PostMarkedReceipt` structure returned when the `IssuePostMarkedReceipt` option is specified and its `Location` element specifies `standalone`. For XMLDSIG-based signatures if the `Location` element specifies `embedded`, then this element will be empty as the entire receipt will be embedded in the XML signature. For PKCS7-based signatures if the `Location` element specifies `embedded`, an RFC 3161 timestamp token will be embedded and this element will also be returned. See **IssuePostMarkedReceipt** above for more details.

SignatureData –When verifying PKCS7-based signatures, if a value of `embedded` is specified in the `Location` element of the `IssuePostMarkedReceipt` The `SignatureData` element will contain an updated PKCS7 signature now containing a signature timestamp embedded in the updated PKCS7 signature as an unsigned attribute. For PKCS7-based signatures which do not request the `embedded` option, this element will be empty.

When using XMLDSIG-based signatures, if a value of `embedded` is specified in the `Location` element of the `IssuePostMarkedReceipt`, then after successful verification, the `IssuePostMarkedReceipt` will be inserted into the incoming signed document and returned to the caller in this element. The `PostMarkedReceipt` will cover all signatures that have been verified. The `NodeName` may affect the scope of the `PostMarkedReceipt` signature, if specified.

ContentMetadata – A string element initialized by the Verify request is returned to the caller.

SignatureInfo – An element whose type is a complex element defined by SignatureInfoType. Please refer to subclause 5.4.4 entitled SignatureInfoType for details.

X509Info – An element whose type is a complex element defined by X509InfoType. Please refer to subclause 5.4.5 entitled X509InfoType for details.

5.5.2 PostMark

5.5.2.1 PostMark Edit Rules Summary

The PostMark operation returns a PostMarkedReceipt structure containing both a timestamp and a receipt. The PostMark can be over any of the 5 content types described below and specified in the `MimeType` attribute of the `Data` input request element. As part of the PostMark operation, the EPM Service will timestamp the content through its TSA or equivalent component and return a PostMarkedReceipt structure containing an RFC 3161-compliant timestamptoken in the `TimeStampToken` element, a `Receipt` structure, and a `ReceiptSignature` structure to the caller. It will be returned in the `PostMarkedReceipt` sub-element of the response. Please refer to subclause 5.4.6 entitled PostMarkedReceipt for details.

5.5.2.2 PostMarkOptions Request Flags

The following option flags can be set in the PostMarkOptionsType complex element. These elements are of type boolean and should be set to ‘True’ or ‘False’, the default value being ‘False’.

```
   <xs:complexType name="PostMarkOptionsType">
      <xs:sequence>
         <xs:element name="EndLifecycle" type="xs:boolean"/>
         <xs:element name="ExtendLifecycle" type="xs:boolean"/>
         <xs:element name="DecryptIncomingEnvelope" type="xs:boolean"/>
```

```
            <xs:element name="EncryptResponse" type="xs:boolean"/>
            <xs:element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
        </xs:sequence>
    </xs:complexType>
```

EndLifecycle – Set to 'True' to end the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

DecryptIncomingEnvelope – Set to 'True' to instruct the EPM Service to decrypt the `Data` element before performing the PostMark operation. This element shall be included as an EnvelopedData object and shall be encrypted with the public key portion of the EPM Service's private decryption key. This option is used to ensure confidential delivery of the transaction content to the EPM Service. See also "EncryptResponse" below.

EncryptResponse – This option requires that the caller sign the request by initializing the `ClaimedIdentity` request element. This option instructs the EPM to encrypt the `SignatureData` element before returning it using the public key present in the incoming `RequesterSignature`. Please also refer to subclause 5.4.12 entitled EncryptResponse Option which covers the use of this option with examples of usage as they apply to the Verify, Decrypt, and RetrieveResults operations.

StoreNonRepudiationEvidence – Set to 'True' to turn on database logging of non-repudiation evidence. If set to 'False', non-repudiation information is not logged to the database. If set to 'False' a TransactionKey is still generated but only skeleton information relating to the transaction status is logged to the database. This skeleton information is logged solely in order to support subsequent retrieval operations and to allow this event to participate in Lifecycles.

5.5.2.3 Postmark Request Elements

Set the following parameters or elements in the PostMarkRequestType complex element before invoking the EPM Service as appropriate:

```
    <xs:element name="PostMarkRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="PostMarkOptions" type="epm:PostMarkOptionsType"/>
                <xs:element name="SignatureType" type="xs:string"/>
                <xs:element name="TransactionKey" type="epm:TransactionKeyType"
nillable="true"/>
                <xs:element name="ClaimedIdentity" type="epm:ClaimedIdentityType"
nillable="true"/>
                <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
                <xs:element name="ClientApplication" type="epm:ClientApplicationType"/>
                <xs:element name="ContentIdentifier" type="xs:string" nillable="true"/>
                <xs:element name="Data" type="epm:QualifiedDataType"/>
                <xs:element name="ContentMetadata" type="epm:ContentMetadataType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

PostMarkOptions – An element whose type is a complex element defined by `PostMarkOptionsType` from the previous subclause.

SignatureType – Specifies the signature type to be used by the EPM when creating the returned `PostMarkedReceipt` structure. Valid values are PKCS7 and XMLDSIG. Although the PostMarkedReceipt structure is itself XML, the internal signature type employed to create the signatures are either PKCS7 or XMLDSIG.

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key and Sequence elements should be initialized as null for PostMark requests unless one is extending a lifecycle. Please refer to TransactionKeyType in subclause 5.4.2.

ClaimedIdentity – The RequesterSignature element of ClaimedIdentity shall be initialized when either Proof-of-Delivery or Proof-of-Possession is desired by the transacting parties, or when this operation is part of a `StartLifecycle` whose `AccessLevel` is `Signed.` (See also

ClaimedIdentity in subclause 5.4.10)

OrganizationID – A string element representing the identifier of the organization requesting the service. This value shall match against a list of valid Organization Identifiers that are registered with the EPM Service.

ClientApplication – See ClientApplication subclause 5.4.17

ContentIdentifier – See ContentIdentifier subclause 5.4.18

Data – A binary element initialized by the caller as a language-specific octet stream containing the content to be time stamped.

MimeType – A string attribute describing the content type of the data contained in the Data element above. Possible values are as follows:

- **Type 1** "text/plain" should be used when passing in plain text data. This data will be hashed (after decoding), for caller convenience prior to internally calling the EPM's timestamping component. A timestamptoken, formatted as per the SignatureType requested, will be returned in the TimeStampToken element of the PostMarkedReceipt. When the SignatureType is XMLDSIG the PostMarkedReceipt signature will be over the content. See Example 2 in the PostMarkedReceipt description above.
- **Type 2** "application/octet-stream" should be used when passing in binary data. This data will also be hashed (after decoding), for the caller's convenience prior to internally calling the EPM's TSA component. A timestamptoken, formatted as per the SignatureType requested, will be returned in the TimeStampToken element of the PostMarkedReceipt. When the SignatureType is XMLDSIG the PostMarkedReceipt signature will be over the content. See Example 2 in the PostMarkedReceipt description.
- **Type 3** "application/vnd.upu.hash-sha1" should be used when the clients themselves are creating the hash of the data. This hash value passed in by the user will be used to construct the call to the timestamping component. The hash algorithm used by the client to create the hash value should be indicated as part of the `MimeType` attribute and should follow the hyphen. Presently only `sha1` and `md5` are accepted. Postal implementations are free to support additional algorithms as required. A `TimeStampToken`, formatted as per the `SignatureType` requested, will be returned in the `TimeStampToken` element of the `PostMarkedReceipt`. When the `SignatureType` is XMLDSIG the `PostMarkedReceipt` signature will be over the content.
- **Type 4** "application/pkcs7-signature" which is signature-specific, indicates that the required timestamp is to be added to the incoming PKCS7 signature structure. The timestamptoken's hash will be calculated over the signature value of the incoming signature. The updated signature now containing an embedded RFC 3161 ASN1 binary timestamptoken will be returned in the response element `SignatureData`. Additionally a `PostMarkedReceipt` structure will also be returned. This is consistent with Verify `IssuePostMarkedReceipt` processing.
- **Type 5** "text/xml", which is signature-specific, indicates that the required `PostMarkedReceipt` is to be calculated over the incoming XMLDSIG-based signature structure. The `PostMarkedReceipt` signature will reference the `SignatureValue` element(s) of the incoming signature(s), and will be returned in the `PostMarkedReceipt` response element.

ContentMetaData – A string element containing custom details of the PostMarked data that can be specified by the client. Example usage could be the original file name, file date, file size or file owner information.

5.5.2.4 PostMark Response Object

A PostMarkResponseType complex element is populated and returned by the EPM Service. Here are the elements it contains:

```
   <xs:element name="PostMarkResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail"
type="epm:TransactionStatusDetailType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
            <xs:element name="PostMarkedReceipt" type="epm:PostMarkedReceiptType"/>
            <xs:element name="SignatureData" type="epm:QualifiedDataType"
nillable="true"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

TransactionStatus – A string element representing the result of the overall transaction. ‘0’ denotes success. ‘1’ denotes that a warning was generated. Transaction failure is denoted by a specific numerical value..

TransactionStatusDetail – An element whose type is a complex element defined by TransactionStatusDetailType. (see TransactionStatus and TransactionStatusDetailType in subclause 5.4.1)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the EPM Service. Please refer to TransactionKeyType in subclause 5.4.2.

PostMarkedReceipt – The `PostMarkedReceipt` element is returned for all PostMark MimeType’s. A standalone `PostMarkedReceipt` of the appropriate type as specified in the `SignatureType` request element, and covering the appropriate content dependent on whether it is a PostMark over data or a PostMark over a signature, will be returned. Please refer to subclause 5.4.6 entitled PostMarkedReceipt for details.

SignatureData – If `MimeType` option is Type 4, then the timestamp will be part of the updated and returned signature structure. For `MimeType` option type 4, the timestamp will be a conventional RFC 3161 binary timestamp token which will be embedded in the incoming PKCS7 signature and returned in this element. The `PostMarkedReceipt,` or the embedded timestamp when using Type 4, is simply an attestation of the existence of a particular piece of data at that moment in time and does not attest to the validity of anything else.

5.5.3 Retrieve Results

5.5.3.1 RetrieveResults Edit Rules Summary

The RetrieveResults operation below is used primarily for accessing evidence required at challenge time. Under this scenario it is executed by the post in support of these evidence requests. It allows the post to retrieve response information from any previous operation identified by the specified `TransactionKey` and `Sequence` qualifier. Therefore it is valid against most operations/verbs (exceptions are: RetrievePostalAttributes, and RetrieveSummary which are already fundamentally retrieval operations and not the subject of disputes.).

The other usage scenario for RetrieveResults is as a document pickup facility. In this use case, a recipient can pickup a document originally placed with the EPM as a result of a sender locally signing a document and subsequently requesting that the EPM Verify that signed document. The act of verifying the document's signature via this origin Verify operation issued by the sender can place both the signature and the signed document (whether embedded or detached) in the EPM's database under a specific TransactionKey, which is returned to the sender. The sender would then pass the TransactionKey “out of band” to the recipient instructing them to “pickup” the document. The recipient would then issue a RetrieveResults operation with this specified TransactionKey, and pickup the document. It should be noted that caution should be taken as to whom is allowed to pickup a document in this fashion. If security is a concern, implementers are encouraged to use a StartLifecycle specifying the `ParticipatingParties` allowed to perform actions against the content. See also `AccessScope` and `AccessLevel` on the StartLifecycle for further access privilege details.

Thus it can be seen that through the use of the `ParticipatingParty` and `ClaimedIdentity` elements, RetrieveResults can also be used as a more powerful "sign for pickup" facility providing end-to-end non-repudiation and Proof-of-Delivery.

EXAMPLE John Smith can setup a Lifecycle by issuing a StartLifecycle operation with `AccessScope` set to `Individual`. He could specify CN=Bill Graham, O=Acme Corporation, C=CA as a valid `ParticipatingParty` and `AccessLevel` for Bill set to `Signed` when issuing the StartLifecycle request. John could also set `NotifyEvents` for Bill to `Verify` so Bill would be eMailed when John posts the document to the EPM via his Verify operation. John then signs the document and subsequently verifies that document/signature using a Verify request to the EPM. This places the verified document and signature along with all results in the EPM database, and triggers a notification to be sent to Bill Graham's eMail address. Bill would then issue a RetrieveResults against John Smith's original TransactionKey in order to "pickup" the signed document. In order to strongly authenticate Bill, before returning the content to him, Bill needs to sign his request, hence "Sign for Pickup". This is accomplished by initializing the `RequesterSignature` element within `ClaimedIdentity`.

This entire sequence can be supported within EPM-enabled desktop applications or can be directly implemented within subscriber EPM-enabled applications . Consult the local Postal administration for details. The qualified form of the TransactionKey including the Sequence element is required in multi-event transaction lifecycles when more than one Verify event exists within the NonRepudiation database under the selected TransactionKey. The Sequence qualifier is used to select which signature verification operation the caller wants the results for. The `RequesterSignature` element of `ClaimedIdentity` shall be initialized.

Public access rules apply to the RetrieveResults to prevent unauthorized access to the operation's potentially sensitive contents. By default, retrieving results against a previous operation can only be accessed by the same individual that initiated that previous operation. In order to allow wider access to the response content of an operation, users needs to set up a `ParticipatingParty` list by issuing a StartLifecycle operation and specifying the desired `AccessScope`, and as many `ParticipatingParty` entries as desired, each with their designated `AccessLevel`. These Lifecycle-specified PartyName(s) and the access rules that govern them, will apply for all operations which are part of this Lifecycle.

NOTE Proof-of-Delivery and Proof-of-Possession can also be accomplished using an origin Verify in conjunction with a destination CheckIntegrity. In this scenario, the signed document is sent directly to the recipient and the EPM is witnessing the origin and destination events. That is, the Verify of the sender's signature over the document, and the recipient upon receipt, passing up that same signed document on a signed CheckIntegrity request. The recipient is asked to sign the `OriginalContent` of the CheckIntegrity request thus ensuring Proof-of-Delivery and Proof-of-Possession.

5.5.3.2 RetrieveResultsOptions Request Flags

The following option flags can be set in the RetrieveResultsOptionsType complex element. These elements are of type boolean and should be set to ‘True’ or ‘False’, the default value being ‘False’. This object will be added as an element reference to the `RetrieveResultsRequestType` element which is referenced in the next subclause.

```
   <xs:complexType name="RetrieveResultsOptionsType">
      <xs:sequence>
         <xs:element name="IssuePostMarkedReceipt"
type="epm:IssuePostMarkedReceiptType" nillable="true"/>
         <xs:element name="EncryptResponse" type="xs:boolean"/>
         <xs:element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
         <xs:element name="ReturnTimeStampAudit" type="xs:boolean"/>
         <xs:element name="ReturnSignatureInfo" type="xs:boolean"/>
         <xs:element name="ReturnX509Info" type="xs:boolean"/>
      </xs:sequence>
   </xs:complexType>
```

IssuePostMarkedReceipt – Set to 'True' in order to cause the EPM to return a receipt attesting to the validity and non-repudiability of the operation. This returned receipt is PostMarked using an EPM-generated timestamptoken. Please refer to subclause 5.4.6 entitled PostMarkedReceipt for details. This token and other receipt information is returned in the `PostMarkedReceipt` element which is itself bound by a signature of authenticity. The `TimeStampToken` within the `PostMarkedReceipt` will be over the `Data` response element returned to the caller. The `embedded` option of the `IssuePostMarkedReceipt` is not supported for RetrieveResults.

EncryptResponse – This option requires that the caller sign the request by initializing the `ClaimedIdentity` request element. This option instructs the EPM to encrypt the `Data` element of the `Results` structure before returning it using the public key present in the incoming `RequesterSignature`. Additionally if the caller has requested that the `SignatureInfo` be returned, then the `SignedContent` element should also be encrypted. Please also refer to subclause 5.4.12 entitled EncryptResponse Option which covers the use of this option with examples of usage as they apply to the Verify, Decrypt, and RetrieveResults operations.

StoreNonRepudiationEvidence – Set to 'True' to turn on database logging of non-repudiation evidence. If set to 'False', non-repudiation information is not logged to the database. If set to 'False' a TransactionKey is still generated but only skeleton information relating to the transaction status is logged to the database. This skeleton information is logged solely in order to support subsequent retrieval operations and to allow this event to participate in Lifecycles.

ReturnTimeStampAudit – Set to 'True' to populate the TimeStampAudit element of the `RetrieveResultsResponseType` in the next subclause.

ReturnSignatureInfo – Set to 'True' to return a detailed breakdown of the signature object, including the original content. Please refer to subclause 5.4.4 entitled SignatureInfoType for details.

ReturnX509Info – Set to 'True' to return a detailed breakdown of the certificate used to perform the operation. Please refer to subclause 5.4.5 entitled X509InfoType for details.

NOTE Since a RetrieveResults is inherently associated with the target `TransactionKey` against which it is operating, EPM implementations shall automatically create, or extend, the Lifecycle associating this operation with the target transaction key in a Lifecycle. For example, if the target operation has `TransactionKey` 1234 with sequence 1, then this RetrieveResults should be assigned `TransactionKey` 1234 with sequence 2. If the target operation is already participating in a Lifecycle, then the EPM implementation shall retain the transaction key and increment the sequence number by 1 for this operation. The same rules apply for CheckIntegrity requests. If the Lifecycle has been explicitly closed, an error should be returned. Applications wishing to continue supporting RetrieveResults and CheckIntegrity against Lifecycles should leave the Lifecycle opened.

5.5.3.3 RetrieveResults Request Elements

Set the following parameters or elements of the RetrieveResultsRequestType object type before invoking the EPM Service as appropriate:

```
   <xs:element name="RetrieveResultsRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="RetrieveResultsOptions" 
type="epm:RetrieveResultsOptionsType"/>
            <xs:element name="SignatureType" type="xs:string"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
            <xs:element name="ClaimedIdentity" type="epm:ClaimedIdentityType" 
nillable="true"/>
            <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
            <xs:element name="ClientApplication" type="epm:ClientApplicationType"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

RetrieveResultsOptions – An element whose type is a complex element defined by `RetrieveResultsOptionsType` from the previous subclause.

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. `Locator` should contain the corresponding EPM application instance reference. `Key` should contain the transaction identifier (key) from the previous target operation one wishes to “retrieve results” and content for. `Sequence` should contain the respective sequence number of the target transaction. Please refer to TransactionKeyType in subclause 5.4.2.

ClaimedIdentity – The RequesterSignature element of ClaimedIdentity shall be initialized when either Proof-of-Delivery or Proof-of-Possession is desired by the transacting parties, or when this operation is part of a `StartLifecycle` whose `AccessLevel` is `Signed.` (See also

ClaimedIdentity in subclause 5.4.10)

SignatureType – Specifies the signature type to be used by the EPM when creating the returned PostMarkedReceipt structure. Valid values are PKCS7 and XMLDSIG.

OrganizationID – A string element representing the identifier of the organization requesting the service. This value shall match against a list of valid Organization Identifiers that are registered with the EPM Service.

ClientApplication – See ClientApplication subclause 5.4.17

5.5.3.4 RetrieveResults Response Object

A RetrieveResultsResponse complex element is populated and returned by the EPM Service. The response covers two distinct categories of information. The first category contains response elements pertaining to the actual execution of RetrieveResults operation itself including the `TransactionStatus`, `TransactionStatusDetail`, the new `TransactionKey` generated for this operation, and an optional `PostMarkedReceipt` if requested. The second category of information returned pertains to the target transaction whose result information is being retrieved. All information pertaining to the retrieved TransactionKey is contained in the `Results` element.
The results for the target `TransactionKey` specified in the request and retrieved from the EPM’s log are populated and returned in the `ResultsType` structure. Description of all response elements follow:

```
   <xs:element name="RetrieveResultsResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail" 
type="epm:TransactionStatusDetailType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
            <xs:element name="PostMarkedReceipt" type="epm:PostMarkedReceiptType"
nillable="true"/>
            <xs:element name="Results" type="epm:ResultsType"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

TransactionStatus – A string element representing the result of the overall transaction. ‘0’ denotes success. ‘1’ denotes that a warning was generated. Transaction failure is denoted by a specific numerical value..

TransactionStatusDetail – An element whose type is a complex element defined by TransactionStatusDetailType. (see TransactionStatus and TransactionStatusDetailType in subclause 5.4.1)

TransactionKey – This is the new TransactionKey generated by the EPM, not to be confused with the target TransactionKey specified in the request and being retrieved. It is a complex type defined by TransactionKeyType. `Locator`, `Key`, and `Sequence` elements will be populated by the EPM Service Please refer to TransactionKeyType in subclause 5.4.2.

PostMarkedReceipt – An optional `PostMarkedReceipt` structure returned when the `IssuePostMarkedReceipt` is specified. This is a receipt issued for having performed the RetrieveResults operation itself and is normally used in “Sign for Pickup” scenarios when Proof-of-Delivery is required. For PKCS7-based signatures, this element is always returned as a standalone receipt when the `IssuePostMarkedReceipt` option is turned on.

NOTE When a `PostMarkedReceipt` is issued for a RetrieveResults operation, it will by definition contain a content timestamp covering the `Data` element of the `ResultsType` which reflects the nature of the target operation (see explanation of the `Data` element below).

ResultsType – This complexType and all its sub-elements contain all information relating to the target transaction being retrieved. Its layout is shown below followed by a description of the associated sub-elements.

```
   <xs:complexType name="ResultsType">
      <xs:sequence>
         <xs:element name="TransactionStatus" type="xs:string"/>
         <xs:element name="TransactionStatusDetail"
type="epm:TransactionStatusDetailType"/>
         <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
         <xs:element name="Operation" type="xs:string"/>
         <xs:element name="OperationOptions" type="epm:ValidOption" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
         <xs:element name="UniqueSequenceId" type="xs:string" nillable="true"/>
         <xs:element name="ClientApplication" type="epm:ClientApplicationType"
nillable="true"/>
         <xs:element name="ContentIdentifier" type="xs:string" nillable="true"/>
         <xs:element name="PostMarkedReceipt" type="epm:PostMarkedReceiptType"
nillable="true"/>
         <xs:element name="Data" type="epm:QualifiedDataType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="ContentMetadata" type="epm:ContentMetadataType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="TimeStampAudit" type="epm:QualifiedDataType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="SignatureInfo" type="epm:SignatureInfoType"
nillable="true"/>
         <xs:element name="X509Info" type="epm:X509InfoType" nillable="true"/>
      </xs:sequence>
   </xs:complexType>
```

TransactionStatus – The TransactionStatus of the original operation being retrieved.

TransactionStatusDetail – The TransactionStatusDetail of the original operation being retrieved. see TransactionStatus and TransactionStatusDetailType in subclause 5.4.1

Operation – The name of operation being retrieved (i.e. Verify, PostMark, CheckIntegrity, etc …)

OperationOptions – Will contain an occurrence for each option that was set to true in the original operation being retrieved.

OrganizationID – The value of the OrganizationID passed in on the original transaction being retrieved.

UniqueSequenceID – Used in conjunction with RetrieveSummary to specify where in the results list we last retrieved result content. The RetrieveSummary operation can start where it last left off using several selector fields as part of the criteria. (refer to the RetrieveSummary in subclause 5.6.6 for details)

ClientApplication – See ClientApplication subclause 5.4.17

ContentIdentifier – See ContentIdentifier subclause 5.4.18

PostMarkedReceipt – This is the PostMarkedReceipt, if any, associated with the target operation being retrieved. Please refer to subclause 5.4.6 entitled PostMarkedReceipt for further details.

Data – This is the contents of the primary element returned for the target operation. It is usually a crypto structure or raw data content. For example, when the previous operation being referred to in the RetrieveResults is a Verify with `IssuePostMarkedReceipt` turned on, then `Data` response element will be initialized with a signature and the `MimeType` will bear the type of signature being returned. This might be a PKCS7 or XMLDSIG signature based on the nature of the previous operation. Similarly for a Sign operation, this `Data` response element would contain the signature created by that operation. In the case of an Encrypt, it would contain the encrypted content from that operation. `Data` could also be initialized with text or binary content.

The table below summarizes what the RetrieveResults `Data` response element of the `ReceiptType` will contain for each target operation type. The element names in the 2nd column are from the response structure for the targeted operation.

Original Operation	Contents of `Data`	Description
CheckIntegrity	**Nill**	
Decrypt	**Data**	The `Data` element will contain the content which was returned on the original Decrypt operation now being retrieved.
Encrypt	**SignatureData**	This element will contain the generated PKCS7 or the XMLDSIG-based `EnvelopedData` structure.
Locate	**PublicEncryptionCert**	The contents of the original encryption certificate returned on the Locate operation.
LogEvent	**Data**	Will contain the content from the `Data` element of the original LogEvent transaction which was passed up to be logged. In this case the retrieved content is actually request data from the target operation.
RetrieveResults	**Nill**	Retrieving the results of a previous RetrieveResults. Nothing will be returned in `Data`.
RetrievePostalAttributes	**Nill**	This operation is already a retrieval and extraction

		operation. Nothing is returned in `Data`.
RetrieveSummary	**Nill**	Same as above.
Sign	**SignatureData**	This element will contain the originally generated PKCS7 or XMLDSIG based signature returned to the caller. *NOTE The `SignedContent` element from the original Sign request is also available in the `SignatureInfo` element of the `ResultsType`.*
StartLifeCycle	**Nill**	
PostMark	**Data**	Will contain the content from the `Data` element of the original PostMark request. In this case the retrieved content is actually request data from the target operation.
Verify	**SignatureData**	If the original Verify request asked for an embedded PostMark, then the `Data` element will contain that verified and PostMarked signature. If the original Verify did not ask for an embedded PostMark, then the `Data` element will contain the originally verified signature. *NOTE The `SignedContent` element from the original Verify request is also available in the `SignatureInfo` element.*

NOTE If the `EncryptResponse` option was set true on any of the original requests being targeted by this RetrieveResults, the appropriate elements will contain an `EnvelopedData` object encrypted for the caller. The `MimeType` and the context will allow the caller to discern which type it is. Please refer to the individual verbs for exactly which elements the `EncryptResponse` applies to.

ContentMetadata – See ContentMetadata

TimeStampAudit – An binary element containing a signed TimeStamp audit log captured from the TSA. Support for the TimeStampAudit element is optional. Provision of this info is at the discretion of each EPM implementation.

SignatureInfoType – An element whose type is a complex element defined by SignatureInfoType. Please refer to subclause 5.4.4 entitled SignatureInfoType for details.

X509InfoType – An element whose type is a complex element defined by X509InfoType. Please refer to subclause 5.4.5 entitled X509InfoType for details.

5.5.4 CheckIntegrity

5.5.4.1 CheckIntegrity Edit Rules Summary

The CheckIntegrity operation allows clients to pass in content from a previous operation (Verify, PostMark, and Sign are valid as previous operations) and have that content byte by byte compared against the original version stored in the EPM's non-repudiation log under the requested TransactionKey. By passing in the TransactionKey of the original operation along with the content to be checked, the EPM will validate whether that content is authentic. A common use case would be as a Proof-of-Delivery, Proof-of-Possession tool whereby a sender can be reassured that the recipient has received the signed document. This is ensured since the recipient not only signs the CheckIntegrity request but also passes in the document received from the sender. In this fashion denial of receipt by the intended recipient cannot be made. It should be noted that this reassurance can really only be obtained when the recipient signs the request using the `RequesterSignature` construct of `ClaimedIdentity`. The `MimeType` attribute of the `OriginalContentType` which specifies the valid values for the type of the eContent being compared is outlined below. See also the description of OriginalContentType above.

Valid values for the `MimeType` attribute of `OriginalContentType` are as follows:

text/plain, application/octet-stream – these attribute values are all valid for comparisons against a previous Sign, Verify, or PostMark operation.

When the CheckIntegrity refers to a previous Sign or Verify operation, the client is expected to pass in the original eContent over which the original signature was either signed or verified. This eContent is then compared against what was previously stored in the EPM's non-repudiation database.

When the CheckIntegrity refers to a previous PostMark operation, the client is expected to pass in the `Data` that was passed in on the original Postmark call and thus is valid only for previous PostMarks of Type 1, Type 2, or Type 3. Types 4 and 5 are signature-based and are not valid and should be rejected with error if this MimeType is being used.

Please refer to the `MimeType` attribute description in subclause 5.5.2.3 entitled Postmark Request Elements for a description of the PostMark Types. `MimeType` text/plain is assumed to be an ASCII representation.

application/vnd.upu-digest-value - the actual hash value over the originally signed eContent is passed in and compared. This `MimeType` is only valid for a CheckIntegrity operation which refers to a previous Sign or Verify operation.

application/timestamp-token - the client is expected to pass in the detached binary RFC 3161 timestamptoken. For example, this would be inside the `TimeStampToken` element of the `PostMarkedReceipt` returned on the referenced Sign, Verify, or PostMark operation.

application/pkcs7-signature - the entire PKCS7 signature shall be passed in and will be compared in its entirety to the original signature from the previous Sign or Verify operation. If the CheckIntegrity refers to a previous PostMark operation, that PostMark operation shall be of Type 4 (i.e. `application/pkcs7-signature).`

text/xml - the contents of the `<dsig:SignatureValue` within the `<dsig:Signature>` structure, without the bounding tags, shall be passed in and will be compared against the equivalent element from the previous Sign, Verify, or PostMark operation. If the CheckIntegrity refers to a previous PostMark operation, that PostMark operation shall be of Type 5 (i.e. `text/xml`).

If senders require Proof of Delivery, implementers should use the CheckIntegrity operation not the Verify operation, as that is the only way to ensure that the specific document (or its hash) was actually received by the recipient. If the recipient is passing in the document on the CheckIntegrity request, as well as signing over that content (See also

ClaimedIdentity in subclause 5.4.10), then the sender can enjoy Proof-of-Delivery, Proof-of-Possession, and Non-Repudiation of receipt. Please also refer to the first use-case scenario for an example of how a CheckIntegrity is used by a recipient to provide Proof-of-Delivery and Proof-of-Possession. Please also refer to subclause 5.4.12 entitled EncryptResponse Option for further details. OriginalContent can be either: content previously signed and verified, a hash value, a timestamptoken, or a digital signature. At verification or challenge time, the user (usually the recipient) can submit multiple pieces of non-repudiation data to be checked for authenticity. It is up to the EPM implementer to check each item and to provide the authenticity results by comparing each item to the content stored in the EPM non-repudiation log. This operation does not perform a cryptographic verification but rather simply compares the `OriginalContent` passed in to that which already exists within the non-repudiation database under the requested `TransactionKey`.

The CheckIntegrity operation requires a valid transaction key (refer to TransactionKeyType in subclause 5.4.2) plus an exact copy of the original content to be checked returned from the original Sign or Verify operation. CheckIntegrity only works on a previous Sign, Verify, or PostMark operation. By referencing the incoming transaction key, the EPM Service will extract the required `MimeType` from the database and its contents will be compared to what was passed in on the request's `OriginalContent` element. For confidentiality purposes, one can encrypt the `OriginalContent` and specify the `DecryptIncomingEnvelope` option to have the EPM Service decrypt the contents before comparing it.

5.5.4.2 CheckIntegrityOptions Request Flags

The following option flags can be set in the CheckIntegrityOptionsType complex element. These elements are of type boolean and should be set to 'True' or 'False', the default value being 'False'. This object will be added as an element reference to the `CheckIntegrityRequestType` complex element referenced in the next subclause.

```
   <xs:complexType name="CheckIntegrityOptionsType">
      <xs:sequence>
         <xs:element name="DecryptIncomingEnvelope" type="xs:boolean"/>
         <xs:element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
         <xs:element name="IssuePostMarkedReceipt"
type="epm:IssuePostMarkedReceiptType" nillable="true"/>
      </xs:sequence>
   </xs:complexType>
```

DecryptIncomingEnvelope – Set to 'True' to request decryption of the incoming OriginalContent element before having it compared to the original data. The enveloped data object shall be encrypted with the public key portion of the EPM Service's private decryption key. Used if this caller wishes to preserve confidentiality when using the CheckIntegrity function. The service will first decrypt the incoming content and compare this decrypted content to the original. The client caller should place the encrypted envelope in the `OriginalContent` field. The service will treat the content according to the `OriginalContentType` specified. If the original Verify did not specify DecryptIncomingEnvelope, the caller (usually the recipient) is probably also not concerned with confidentiality and therefore need not turn on the DecryptIncomingEnvelope flag. The caller should specify the OriginalContent's `MimeType` attribute, that is the data that was originally signed, as simply `Data` in the OriginalContent's `MimeType`. The public encryption key used shall be the Postal Administrations public key.

StoreNonRepudiationEvidence – On a CheckIntegrity logging of information in the non-repudiation database may be required by a customer to attest to the fact that the intended recipient has both received the original signed document and has requested confirmation as to its authenticity. This captured evidence can be used to support non-repudiation of delivery, receipt, and knowledge. It is also assumed that the sender delivers the TransactionKey only to the intended recipients. If this is deemed insufficiently secure, customers should use the ParticipatingParty facility described above in order to restrict access only to the intended recipients.

IssuePostMarkedReceipt – Set to `True` in order to cause the EPM to return a receipt attesting to the validity and non-repudiability of the operation. This option will only be honored if the `RequesterSignature` has been provided on the request. See also

ClaimedIdentity in subclause 5.4.10. This returned receipt is PostMarked using an EPM-generated timestamptoken. Please refer to subclause 5.4.6 entitled PostMarkedReceipt for details. This token and other receipt information is returned in the PostMarkedReceipt element which is itself bound by a signature of authenticity. The EPM will create the `PostMarkedReceipt` signature over the `OriginalContent` occurrences within the request. The `embedded` option of the `IssuePostMarkedReceipt` is not supported for CheckIntegrity.

NOTE Since a CheckIntegrity is inherently associated with the target TransactionKey against which it is operating, EPM implementations shall automatically create, or extend, the Lifecycle associating this operation with the target transaction key in a Lifecycle. For example, if the target operation has transaction key 1234 with sequence 1, then this CheckIntegrity should be assigned transaction key 1234 with sequence 2. If the target operation is already participating in a Lifecycle, then the EPM implementation shall retain the transaction key and increment the sequence number by 1 for this operation. The same rules apply for RetrieveResults requests. If the Lifecycle has been explicitly closed, *an error should be returned. Applications wishing to continue supporting RetrieveResults and CheckIntegrity against Lifecycles should leave the Lifecycle opened*

5.5.4.3 CheckIntegrity Request Elements

Set the following parameters or elements of the CheckIntegrityRequestType complex element before invoking the EPM Service as appropriate:

```
   <xs:element name="CheckIntegrityRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="CheckIntegrityOptions"
type="epm:CheckIntegrityOptionsType"/>
            <xs:element name="SignatureType" type="xs:string" nillable="true"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
            <xs:element name="ClaimedIdentity" type="epm:ClaimedIdentityType"
nillable="true"/>
            <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
            <xs:element name="ClientApplication" type="epm:ClientApplicationType"/>
            <xs:element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <xs:element name="OriginalContent" type="epm:OriginalContentType"
minOccurs="1" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

CheckIntegrityOptions – An element whose type is a complex element defined by CheckIntegrityOptionsType from the previous subclause.

TransactionKey– An element whose type is a complex element defined by TransactionKeyType. Locator should contain the corresponding application instance reference. Key should contain the transaction ID/key from the previous sign or verify you wish to compare. Sequence should contain the respective sequence number of the previous Sign or Verify. Please refer to TransactionKeyType in subclause 5.4.2.

OriginalContent – This is the content to be checked against the target operation. The `OriginalContent` element shall always be initialized. For instance, when a previous Verify is the target `TransactionKey` of this CheckIntegrity request, one could pass the signed content of that previous Verify operation in to be checked against the copy of that signed content stored within the EPM's non-repudiation database under that `TransactionKey`. If the values compare equally, the CheckIntegrity will return success. In this fashion, for example, recipients of signed documents could check whether document signatures were in fact legitimate and whether the document they have in their possession is in fact the exact same as the original document that was signed by the sender. Please also refer to subclause 5.4.8 entitled OriginalContentType for more details.

ClaimedIdentity – The RequesterSignature element of ClaimedIdentity shall be initialized when either Proof-of-Delivery or Proof-of-Possession is desired by the transacting parties, or when this operation is part of a `StartLifecycle` whose `AccessLevel` is `Signed` as previously discussed. In the Proof-of-Delivery or Proof-of-Possession case, the `RequesterSignature` shall be a signature over the hash of the `OriginalContent` element whose integrity is being checked. In this fashion the requester of the operation cannot deny having both received and been in possession of that content. The data over which this signature is created should be inclusive of all XML element tags as well.

Example eContent to the PKCS7 signature creation is shown below:

EXAMPLE 1

```
<OriginalContent MimeType="application/pkcs7-signature">MIIDAYhvcNAQcC
...</OriginalContent>
```

If there is more than one `OriginalContent` occurrence, since it is an "unbounded" element, the above structure would be repeated for each `OriginalContent` element to be checked, and all needs to be input to the signature creation. Please also refer to the Serialization Conventions in subclause 5.4.10. The `MimeType` attribute should be left in to differentiate the `OriginalContent` elements of which there can be more than one.

EXAMPLE 2

```
<OriginalContent MimeType="application/pkcs7-signature">MIIhvcANcC
...</OriginalContent>
<OriginalContent MimeType="application/timestamp-token">MIIDAYhcvNA
...</OriginalContent>
```

When Proof-of-Delivery or Proof-of-Possession is required, and the SignatureType is XMLDSIG, the RequesterSignature should be initialized. This is shown in Example 5 – RequesterSignature over OriginalContent when used in a CheckIntegrity operation. Please note that in order to avoid duplicating the `OriginalContent` payload in both the `OriginalContent` element as well as within the enveloping `RequesterSignature` structure, callers need to hash the `OriginalContent` value(s) and perform the sign operation over the resultant hash value of the `OriginalContent` occurrences. Proof-of-Possession can be verified by the EPM implementation by re-deriving the hash over the request's `OriginalContent` element, comparing this re-derived hash value against the element contents of the signed content (2nd line in Example 5 – RequesterSignature over OriginalContent when used in a CheckIntegrity operation and finally cryptographically verifying the RequesterSignature as presented in the request.

If this CheckIntegrity is within a Lifecycle whose AccessLevel is set to Signed, and Proof-of-Delivery is required, then this `RequesterSignature` over the hash of the `OriginalContent` is used instead of over the `TransactionKey` as the signature granting access to this Lifecycle. This removes the need for callers to sign 2 separate pieces of information. EPM implementations would still use the identity information in the public verification key in the signature to compare against the `ParticipatingParty` list specified in the Lifecycle to ascertain access privilege. For more details, please also refer to subclause 5.6.5 entitled StartLifeCycle.

SignatureType – Specifies the signature type to be used by the EPM when creating the returned PostMarkedReceipt structure. Valid values are PKCS7 and XMLDSIG.

OrganizationID – A string element representing the identifier of the organization requesting the service. This value must match against a list of valid Organization Identifiers that are registered with the EPM Service.

ClientApplication – See ClientApplication subclause 5.4.17

5.5.4.4 CheckIntegrity Response Object

A CheckIntegrityResponseType complex element is populated and returned by the EPM Service. Here are the elements it contains:

```
   <xs:element name="CheckIntegrityResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail"
type="epm:TransactionStatusDetailType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
            <xs:element name="PostMarkedReceipt" type="epm:PostMarkedReceiptType"
nillable="true"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

TransactionStatus – A string element representing the result of the overall transaction. ‘0’ denotes success. ‘1’ denotes that a warning was generated. Transaction failure is denoted by a specific numerical value..

TransactionStatusDetail – An element whose type is a complex element defined by TransactionStatusDetailType. (see TransactionStatus and TransactionStatusDetailType in subclause 5.4.1)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the EPM Service. Please refer to TransactionKeyType in subclause 5.4.2.

PostMarkedReceipt – This structure is returned to the caller if they requested the IssuePostMarkedReceipt option on the request. The TimeStampToken within the PostMarkedReceipt will be over the OriginalContent passed in on the request.

The table below summarizes what is required as input to the CheckIntegrity and describes the valid MimeType for each target operation being checked.

Target Operation being checked	**`OriginalContent's MimeType`**	**Processing Rules for `OriginalContent`**
Verify	**text/plain**	Caller shall initialize the `OriginalContent` element with the `SignedContent` over which the original signature was created and subsequently verified. This plain text string was optionally returned on the target Verify call in the `SignedContent` element of `SignatureInfo`. It is the eContent of the signature and will be compared against that which was stored in the non-repudiation log as a result of the Verify operation.
	application/octet-stream	Same as above except that the `SignedContent` to be compared is binary.
	application/vnd.upu-digest-value	Caller shall initialize the `OriginalContent` element with the `ContentHash` produced as a result of signature creation. This hash value was optionally returned on the Verify call in the `ContentHash` element of `SignatureInfo`. This hash value will be compared against that which was stored in the non-repudiation log as a result of the Verify operation. *NOTE Do not base64 encode the `ContentHash` twice when initializing `OriginalContent`.*
	application/timestamp-token	Caller shall initialize the `OriginalContent` element with the detached binary RFC 3161 `TimeStampToken` returned on the Verify call. This token would be inside the `TimeStampToken` element of the `PostMarkedReceipt` originally returned on the target Verify operation. The original Verify operation needs to have specified the `IssuePostMarkedReceipt` option in order for this CheckIntegrity to return success.
	application/pkcs7-signature	Caller shall initialize the `OriginalContent`

		element with the contents of the `SignatureData` element containing the PKCS7 signature that was originally verified. This signature will be compared byte by byte against that which was stored in the non-repudiation log for the original Verify operation.
	text/xml	Caller shall initialize the `OriginalContent` element with the contents of the `<dsig:SignatureValue>` within the `<dsig:Signature>` structure originally Verified, without the bounding tags. This value will be compared against the equivalent element from the target Verify operation. *NOTE Do not base64 encode the `SignatureValue` twice when initializing `OriginalContent`.*
Sign	**text/plain**	Caller shall initialize the `OriginalContent` element with the `SignedContent` over which the original signature was created. This plain text string was passed in to be signed by the original Sign operation. It is the eContent of the signature and will be compared against that which was stored in the non-repudiation log as a result of the Sign operation.
	application/octet-stream	Same as above except that the `SignedContent` to be compared is binary.
	application/vnd.upu-digest-value	Caller shall initialize the `OriginalContent` element with the `ContentHash` produced as a result of the original signature creation. This hash value was optionally returned on the Sign call in the `ContentHash` element of `SignatureInfo`. This hash value will be compared against that which was stored in the non-repudiation log as a result of the Sign operation. *NOTE Do not base64 encode the `ContentHash` twice when initializing `OriginalContent`.*
	application/timestamp-token	Caller shall initialize the `OriginalContent` element with the detached binary RFC 3161 `TimeStampToken` returned on the Sign call. This token would be inside the `TimeStampToken` element of the `PostMarkedReceipt` originally returned on the target Sign operation. The original Sign operation needs to have specified the `IssuePostMarkedReceipt` option in order for this CheckIntegrity to return success.

	application/pkcs7-signature	Caller shall initialize the `OriginalContent` element with the contents of the `SignatureData` element containing the PKCS7 signature that was originally created by the Sign operation. This signature will be compared byte by byte against that which was stored in the non-repudiation log for the original Sign operation.
	text/xml	Caller shall initialize the `OriginalContent` element with the contents of the `<dsig:SignatureValue>` within the `<dsig:Signature>` structure originally Signed, without the bounding tags. This value will be compared against the equivalent element from the target Sign operation. *NOTE Do not base64 encode the `SignatureValue` twice when initializing `OriginalContent`.*
PostMark	**text/plain**	Caller shall initialize the `OriginalContent` element with the contents of the `Data` element passed in on the target PostMark operation being checked. This plain text string was passed in to be timestamped by the original PostMark operation. This content will be compared against that which was stored in the non-repudiation log as a result of the PostMark operation. The original PostMark operation needs to be a Type 1 PostMark to use this `MimeType`.
	application/octet-stream	Caller shall initialize the `OriginalContent` element with the contents of the `Data` element passed in on the target PostMark operation being checked. This binary content was passed in to be timestamped by the original PostMark operation. This content will be compared against that which was stored in the non-repudiation log as a result of the PostMark operation. The original PostMark operation needs to be a Type 2 PostMark to use this `MimeType`.
	application/vnd.upu-digest-value	Not a valid MimeType for a CheckIntegrity against a previous PostMark. Use the `application/timestamp-token` MimeType below for token-related comparisons.
	application/timestamp-token	Caller shall initialize the `OriginalContent` element with the detached binary RFC 3161 `TimeStampToken` returned on the PostMark call. This token would be inside the `TimeStampToken` element of the `PostMarkedReceipt` originally returned on the target PostMark operation.

	application/pkcs7-signature	Not a valid MimeType for a CheckIntegrity against a previous PostMark. Use the `application/timestamp-token` for signature comparisons.
	text/xml	Caller shall initialize the `OriginalContent` element with the contents of the `<dsig:SignatureValue>` within the `<dsig:Signature>` structure of the original PostMark, without the bounding tags. This value will be compared against the equivalent element from the target PostMark operation. *NOTE Do not base64 encode the `SignatureValue` twice when initializing `OriginalContent`.*

5.5.5 Sign

5.5.5.1 Sign Edit Rules Summary

It should be mentioned that the Sign operation is a server-side Sign and not the Sign performed by the client endpoint (normally a customer or ISV application). The Sign is useful under 2 main use-cases. The first special Sign use-case is used when subscribing organizations, wishing to ensure their partners that they are in fact receiving content that originated from them, sign outgoing content with a “Corporate Seal”. The second use-case is a “Server-Side Delegated Signing” operation where the key-pair and certificate resides either with the EPM Service or is delegated to the enterprise or an Identity Service Provider working in conjunction with the customer and the Postal Administration. This alleviates the Postal Administration of the administrative burden of distributing PKI certificate credentials to user end-points.

The more conventional Sign usage scenario is when a desktop application signs content with any CMS/PKCS7 or XMLDSIG compliant crypto library and then subsequently passes the signature object to the EPM for verification and PostMarking using the EPM’s Verify operation.

The Sign operation requires content to be signed and returns it in a PKCS7 or XMLDSIG signature object. ‘Signing’ is the conventional and well understood process whereby a hash representation of the content is encrypted with the private signing key of the signer and converted into a Signature object of the appropriate type.

*NOTE EPM implementations are free to choose the signature algorithm used when performing delegated signing using this operation. However the `sha1WithRSAEncryption` algorithm **SHALL** at a minimum be supported by EPM implementations when creating and verifying signatures of all types including those created by Sign requests as in this subclause. For XMLDSIG-templates which might specify a signing algorithm in the SignatureMethod element, EPM implementations should honor or reject requests based on their capability.*

5.5.5.2 SignOptions Request Flags

The following option flags can be set in the SignOptionsType complex element. These elements are of type boolean and should be set to ‘True’ or ‘False’, the default value being ‘False’. This object will be added as an element reference to the `SignRequestType` complex element which is referenced in the next subclause.

```
   <xs:complexType name="SignOptionsType">
```

```
       <xs:sequence>
          <xs:element name="EndLifecycle" type="xs:boolean"/>
          <xs:element name="ExtendLifecycle" type="xs:boolean"/>
          <xs:element name="VerifyCertificate" type="xs:boolean"/>
          <xs:element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
          <xs:element name="IssuePostMarkedReceipt"
type="epm:IssuePostMarkedReceiptType" nillable="true"/>
          <xs:element name="DecryptIncomingEnvelope" type="xs:boolean"/>
          <xs:element name="EncryptResponse" type="xs:boolean"/>
          <xs:element name="ReturnSignatureInfo" type="xs:boolean"/>
          <xs:element name="ReturnX509Info" type="xs:boolean"/>
       </xs:sequence>
    </xs:complexType>
```

EndLifecycle – Set to 'True' to end the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

VerifyCertificate – This option is most useful in a server-side signing scenario where keying material is held by the post or a party working with the post, and allows the calling application to have the revocation status of the certificate about to be used for this signing operation to be checked via either OCSP or CRL checking.

StoreNonRepudiationEvidence – Set to 'True' to turn on database logging of non-repudiation evidence. If set to 'False', non-repudiation information is not logged to the database. If set to 'False' a TransactionKey is still generated but only skeleton information relating to the transaction status is logged to the database. This skeleton information is logged solely in order to support subsequent retrieval operations and to allow this event to participate in Lifecycles.

IssuePostMarkedReceipt – Set to 'True' in order to cause the EPM to return a receipt attesting to the validity and non-repudiability of the operation. This returned receipt is PostMarked using an EPM-generated timestamptoken. Please refer to subclause 5.4.6 entitled PostMarkedReceipt for details. This token and other receipt information is returned in the `PostMarkedReceipt` element which is itself bound by a signature of authenticity. It should be noted that the `TimeStampToken` contained in the returned `PostMarkedReceipt` is a signature timestamp calculated over the `SignatureValue` (in the case of XMLDSIG), or the `PKCS1` (in the case of PKCS7) signatures.

DecryptIncomingEnvelope – Set to 'True' to instruct the EPM Service to decrypt the `Data` element. Before performing the Sign operation. This element shall be included as an EnvelopedData object and shall be encrypted with the public key portion of the EPM Service's private decryption key. This option is used to ensure confidential delivery of the transaction content to the EPM Service. See also "EncryptResponse" below.

EncryptResponse – This option requires that the caller sign the request by initializing the `ClaimedIdentity` request element. This option instructs the EPM to encrypt the `SignatureData` element before returning it using the public key present in the incoming `RequesterSignature`. Additionally if the caller has requested that the `SignatureInfo` be returned, then the `SignedContent` sub-element should also be encrypted. Please also refer to subclause 5.4.12 entitled EncryptResponse Option which covers the use of this option with examples of usage as they apply to the Verify, Decrypt, and RetrieveResults operations.

ReturnSignatureInfo – Set to 'True' to return a detailed breakdown of the signature object, including the original content. Please refer to subclause 5.4.4 entitled SignatureInfoType for details.

ReturnX509Info – Set to 'True' to return a detailed breakdown of the certificate used to perform the operation. Please refer to subclause 5.4.5 entitled X509InfoType for details.

5.5.5.3 Sign Request Elements

Set the following parameters or elements of the SignRequestType complex element before invoking the EPM Service as appropriate:

```
   <xs:element name="SignRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="SignOptions" type="epm:SignOptionsType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"
nillable="true"/>
            <xs:element name="ClaimedIdentity" type="epm:ClaimedIdentityType"
nillable="true"/>
            <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
            <xs:element name="ClientApplication" type="epm:ClientApplicationType"/>
            <xs:element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <xs:element name="Data" type="epm:QualifiedDataType"/>
            <xs:element name="SignatureType" type="xs:string"/>
            <xs:element name="KeyName" type="xs:string" nillable="true"/>
            <xs:element name="SignaturePolicyID" type="xs:anyURI" nillable="true"/>
            <xs:element name="ContentMetadata" type="epm:ContentMetadataType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

SignOptions – An element whose type is a complex element defined by `SignOptionsType` from the previous subclause.

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key and Sequence elements should be initialized as null for Sign requests unless one is extending a lifecycle. Please refer to TransactionKeyType in subclause 5.4.2.

ClaimedIdentity – The `RequesterSignature` element of `ClaimedIdentity` shall be initialized when this operation is part of a `StartLifecycle` whose `AccessLevel` is `Signed.` See also

ClaimedIdentity in subclause 5.4.10

OrganizationID – A string element representing the identifier of the organization requesting the service. This value must match against a list of valid Organization Identifiers that are registered with the EPM Service.

ClientApplication – See ClientApplication subclause 5.4.17

ContentIdentifier – See ContentIdentifier subclause 5.4.18

Data – The Data element needs to always be base64Binary even if it is text. The MimeType attribute is simply used to tell the EPM that after base64 decoding, this Data element contains a specific `MimeType` and instructs the EPM how to format the input Data in the resulting signature. Valid values of `MimeType` are `text/xml`, `text/plain`, or `application/octet-stream`.

MimeType – This is an attribute of the `Data` element above. Valid values are `text/xml`, `text/plain`, or `application/octet-stream`. The `MimeType` specified also controls the formatting of the "data to be signed" as it will exist *within* the signature. `MimeType` values of either `text/xml` or `text/plain` result in signatures with text representations within their respective signature structures whether they be XMLDSIG or PKCS7. A `MimeType` value of `application/octet-stream` results in binary SignedData content if `SignatureType` is PKCS7 or an `xmldsig#base64` transform if `SignatureType` is XMLDSIG.

SignatureType – A string element containing the desired format of the signature to be returned. This is the desired format of the signature to be created and returned. Valid values are PKCS7, PKCS7-detached, XMLDSIG, XMLDSIG-enveloping, XMLDSIG-detached, and XMLDSIG-template. If XMLDSIG is specified, a simple enveloped XML Digital Signature will be returned, which is the default for XMLDSIG signatures. Users who wish to have the signature formatted as an enveloping signature should specify XMLDSIG-enveloping. Similarly users who wish to have the signature formatted as a detached signature should specify XMLDSIG-detached. Specifying a value of XMLDSIG-template instructs the EPM to treat the incoming `Data` element as a signing template which will be used to construct the resulting XMLDSIG signature as dictated by the template. Please refer to Annex B (Informative) Examples for specific examples of signing templates.

KeyName – This optional element is used in server-side signing scenarios where an EPM is responsible for the maintenance of client key material including both public and private keys. This element, which is synonymous with the XMLDSIG X509SubjectName or Distinguished Name, is used to specify the key to use for this sign operation. Example:

C=CA, S=Ontario, L=Ottawa, O=Acme Corporation, OU=Customer Services, CN=Ed Smith, E=ed.smith@yahoo.ca

This string, or some derivative of it, is used by the EPM implementation to access the private key for this sign operation. EPM implementations are free to decide how authentication and the password is utilized in server-side signing scenarios. The suggested approach is to have the password specified by the user for "Authorization: Basic" in the HTTP header as the password for opening the private key. Anonymous UserIDs should not be permitted in this scenario. Optionally, the EPM implementation may use the `AlternateIdentity` element of the `SupportingInfoType` for more specialized requirements.

SignaturePolicyID –This optional element can be used in server-side signing scenarios and allows the requester to specify the inclusion of a signature policy identifier in the signature's scope. The `SignaturePolicyID` is an abstract type and allows individual Posts to institute whatever their local jurisdiction's mandate or legislate. There is an example of a Signature Policy structure in subclause 5.4.20 entitled SignaturePolicyIdentifier.

ContentMetaData – A string element containing custom details of the signed data that can be specified by the client. Example usage could be the original file name, file date, file size or file owner information.

5.5.5.4 Sign Response Object

A SignResponseType complex element is populated and returned by the EPM Service. Here are the elements it contains:

```
   <xs:element name="SignResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail"
type="epm:TransactionStatusDetailType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
            <xs:element name="PostMarkedReceipt" type="epm:PostMarkedReceiptType"
nillable="true"/>
            <xs:element name="SignatureData" type="epm:QualifiedDataType"
nillable="true"/>
            <xs:element name="SignatureInfo" type="epm:SignatureInfoType"
nillable="true"/>
            <xs:element name="X509Info" type="epm:X509InfoType" nillable="true"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

TransactionStatus – A string element representing the result of the overall transaction. '0' denotes success. '1' denotes that a warning was generated. Transaction failure is denoted by a specific numerical value..

TransactionStatusDetail – An element whose type is a complex element defined by

TransactionStatusDetailType. (see TransactionStatus and TransactionStatusDetailType in subclause 5.4.1)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the EPM Service. Please refer to TransactionKeyType in subclause 5.4.2.

PostMarkedReceipt – An optional `PostMarkedReceipt` structure is returned when the `IssuePostMarkedReceipt` option is specified on the request. For PKCS7-based signatures, this element is always returned when the `IssuePostMarkedReceipt` option is turned on.

For XMLDSIG-based signatures, the PostMarkedReceipt will be embedded in the signed document generated as part of this operation. The `PostMarkedReceipt` will be returned in the `SignatureData` element described below. Please refer to subclause 5.4.6 entitled PostMarkedReceipt for details.

SignatureData –This element will contain the generated PKCS7 or XMLDSIG based signature.

NOTE If the `Location` sub-element of the `IssuePostMarkedReceipt` option specifies `embedded`, then this element will contain either the signed and now Postmarked XML document (XMLDSIG), or the PKCS7 signature with an embedded timestamp included as an unsigned attribute. See subclause 5.4.6 PostMarkedReceipt for more details.

SignatureInfoType – An element whose type is a complex element defined by SignatureInfoType Please refer to subclause 5.4.4 entitled SignatureInfoType for details.

X509InfoType – An element whose type is a complex element defined by X509InfoType. Please refer to subclause 5.4.5 entitled X509InfoType for details.

5.6 EPM Interface Specification – "Extended" Operations in Detail

The following operations: Encrypt, Decrypt, Locate, StartLifecycle, LogEvent, RetrieveSummary, and RetrievePostalAttributes have been identified as extended UPU supported services. The following is a description of each of these operations and how to process them in conjunction with an EPM Web-based service.

5.6.1 Encrypt

5.6.1.1 Encrypt Edit Rules Summary

The Encrypt operation encrypts and returns an enveloped data object of the desired format. Encrypting is the process whereby a random symmetric key is chosen (e.g. 3DES) to encrypt the original data. This symmetric key is subsequently encrypted (RSA key transport) using the recipient's public key, then formatted as either a PKCS7 enveloped data object or an XML Encryption compliant XML `EncryptedData` node. he Encrypt operation will encrypt any incoming data with a certificate specified by the client, and return the encrypted envelope.

The Encrypt operation provides the native EPMs confidentiality support. The EPM Service additionally supports the ability to retrieve the public encryption certificate to be used for the encrypt operation by means of the CertificateID parameter described below. This service is normally used by organizational subscribers wishing to encrypt content for parties they are dealing with. This operation could be appropriate for organizations wishing to encrypt content for their partners or customers in an eBusiness scenario.

There is a special case optional usage of the EPMs confidentiality capability that works in conjunction with a local encrypt operation issued by an individual client customer using the EPM's public encryption key. It is termed "Delegated Confidentiality" within the EPM context. This capability frees individuals from having to manage the public keys of intended recipients. When a subscriber wishes to encrypt content for confidentiality reasons, they simply encrypt that content locally with a public key provided to them by the Postal Administration. This single public key belongs to the post and is the only public key the customer needs to maintain on their desktop or within their application. This local encryption using the post-specific public key can be used with any EPM operation supporting the `DecryptIncomingEnvelope` option. After having encrypted the content with this post-specific public key, the content is now secured for transport. This envelope can be sent to the recipient.

At the recipients end, the recipient does not have the private key required to decrypt this envelope they just received. Consequently they needs to ask the post's EPM Service to Decrypt it for them. It would not make sense for the EPM to simply Decrypt the content and pass it back to the caller in the clear. To prevent this exposure, the EPM can “encrypt the response”. This is activated when the `EncryptResponse` option is turned on in the recipient's Decrypt (or Verify) request. The EPM however requires the callers public key in order to be able to actually encrypt the response prior to returning it (to the recipient in this scenario). In order to provide the EPM with the callers public key, the caller signs the Decrypt (or Verify) request. Since a signature contains the public verification key, this key can then be used to “encrypt the response” for the caller.

Similar scenarios can be exercised where the recipient 1) utilizes a CheckIntegrity operation instead of a Verify, or 2) utilizes a signed RetrieveResults to pick up a document signed by an originator which was left with the EPM. This usage is termed “Sign for Pickup” which may also optionally employ the `EncryptResponse` option, and avoids the need for the sender to transport the document over the public Internet.

Please also refer to subclause 5.4.12 entitled EncryptResponse Option for a detailed explanation which covers the use of this option with examples of usage as they apply to the Verify, Decrypt, and RetrieveResults operations.

5.6.1.2 EncryptOptions Request Flags

The following option flags can be set in the EncryptOptionsType complex element. These elements are of type boolean and should be set to ‘True’ or ‘False’, the default value being ‘False’. This object will be added as an element to the `EncryptRequestType` complex element which is referenced in the next subclause.

```
   <xs:complexType name="EncryptOptionsType">
      <xs:sequence>
         <xs:element name="EndLifecycle" type="xs:boolean"/>
         <xs:element name="ExtendLifecycle" type="xs:boolean"/>
         <xs:element name="VerifyCertificate" type="xs:boolean"/>
         <xs:element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
         <xs:element name="IssuePostMarkedReceipt"
type="epm:IssuePostMarkedReceiptType" nillable="true"/>
         <xs:element name="DecryptIncomingEnvelope" type="xs:boolean"/>
         <xs:element name="ReturnSignatureInfo" type="xs:boolean"/>
         <xs:element name="ReturnX509Info" type="xs:boolean"/>
      </xs:sequence>
   </xs:complexType>
```

EndLifecycle – Set to ‘True’ to end the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

ExtendLifecycle – Set to ‘True’ to extend the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

VerifyCertificate – Set to ‘True’ to enable certificate status checking. If set to ‘False’, the certificate status checking will be bypassed, but the standard certificate validation will still occur. All certificate validation results will be returned in the X509InfoType complex element Please refer to subclause 5.4.5 entitled X509InfoType for details..

StoreNonRepudiationEvidence – Set to ‘True’ to turn on database logging of non-repudiation evidence. If set to ‘False’, non-repudiation information is not logged to the database. If set to ‘False’ a TransactionKey is still generated but only skeleton information relating to the transaction status is logged to the database. This skeleton information is logged solely in order to support subsequent retrieval operations and to allow this event to participate in Lifecycles.

IssuePostMarkedReceipt – Set to 'True' in order to cause the EPM to return a receipt attesting to the validity and non-repudiability of the operation. The input to the timestamp operation included in this receipt will be the contents of the `Data` element passed in on the request. This returned receipt is PostMarked and includes an EPM-generated timestamptoken. Please refer to subclause 5.4.6 entitled PostMarkedReceipt for details. This token and other receipt information is returned in the `PostMarkedReceipt` element which is itself bound by a signature of authenticity.

DecryptIncomingEnvelope – Set to 'True' to instruct the EPM Service to decrypt the `Data` element before performing the Encrypt operation. This element needs to be included as an EnvelopedData object and needs to be encrypted with the public key portion of the EPM Service's private decryption key. This option is used to ensure confidential delivery of the transaction content to the EPM Service prior to being actually encrypted for the final intended recipient. See also `CertificateID` below. Once the `Data` has been received by the EPM and decrypted it can then be encrypted for its final intended recipient(s). When used in this fashion, the EPM is essentially performing a "Re-Encrypt".

ReturnSignatureInfo – Set to 'True' to return a detailed breakdown of the signature object, including the original content. Please refer to subclause 5.4.4 entitled SignatureInfoType for details.

ReturnX509Info – a Set to 'True' to return a detailed breakdown of the certificate used to perform the operation. Please refer to subclause 5.4.5 entitled X509InfoType for details.

5.6.1.3 Encrypt Request Elements

Set the following parameters or elements of the EncryptRequestType complex element before invoking the EPM Service as appropriate:

```
   <xs:element name="EncryptRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="EncryptOptions" type="epm:EncryptOptionsType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType" 
nillable="true"/>
            <xs:element name="ClaimedIdentity" type="epm:ClaimedIdentityType" 
nillable="true"/>
            <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
            <xs:element name="ClientApplication" type="epm:ClientApplicationType"/>
            <xs:element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <xs:element name="Data" type="epm:QualifiedDataType"/>
            <xs:element name="SignatureType" type="xs:string"/>
            <xs:element name="NodeName" type="xs:string" nillable="true"/>
            <xs:element name="SessionKeyAlgo" type="xs:string" nillable="true"/>
            <xs:element name="CertificateSearchType" type="xs:string"/>
            <xs:element name="CertificateID" type="xs:string" minOccurs="1" 
maxOccurs="unbounded"/>
            <xs:element name="ContentMetadata" type="epm:ContentMetadataType" 
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

EncryptOptions – An element whose type is a complex element defined by `EncryptOptionsType` from the previous subclause.

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key and Sequence elements should be initialized as 'null' for Encrypt requests unless one is extending a lifecycle. Please refer to TransactionKeyType in subclause 5.4.2.

ClaimedIdentity – The RequesterSignature element of ClaimedIdentity shall be initialized when either

Proof-of-Delivery or Proof-of-Possession is desired by the transacting parties, or when this operation is part of a `StartLifecycle` whose `AccessLevel` is `Signed`. See also

ClaimedIdentity in subclause 5.4.10.

OrganizationID – A string element representing the identifier of the organization requesting the service. This value must match against a list of valid Organization Identifiers that are registered with the EPM Service.

ClientApplication – See ClientApplication subclause 5.4.17

Data – The Data element shall always be base64Binary even if it is `text/xml`. The `MimeType` attribute is simply used to tell the EPM that after decoding, this Data element contains a specific `MimeType` and instructs the EPM how to handle the input `Data`. Valid values of `MimeType` are `text/xml`, `text/plain`, or `application/octet-stream`.

MimeType – This is an attribute of the Data element above. Valid values of `MimeType` are `text/xml`, `text/plain`, or `application/octet-stream`.

SignatureType – A string element containing the desired format of the envelope to be returned. This is the desired format of the envelope to be returned. Valid values are PKCS7, XMLDSIG, and XMLDSIG-template. If XMLDSIG is specified, a simple XML document supporting the XML Encryption standard will be returned. If XMLDSIG-template is specified, the EPM will construct the XML Encryption compliant document based on the template passed in.

NodeName – Used with XMLDSIG types, this element specifies which node in the XML document to encrypt. This element's contents will be replaced by an `EnvelopedData` XML Encryption compliant node. Expressed as a string, a namespace URI qualifier may precede the actual node name, e.g. "pay:Salary". If omitted, the root node of the document passed in `Data` will be encrypted.

SessionKeyAlgo – EPM implementations **SHALL**, at a minimum, support the wrapping of session content encryption keys with RSA asymmetric encryption for purposes of Key Transport as per PKCS#7 1.5. That is to say that the Key Encryption Algorithm Identifier is rsaEncryption. This ensures maximum compatibility with legacy PKCS7 libraries and allows certificate-based public key identification of recipients. This element indicates the encryption class and size to be used to compute the session key. It will be used for signatures of both types i.e. PKCS7 and XMLDSIG. Valid values are formatted as a `class-size` string, e.g. `aes128-cbc`, `aes192-cbc`, `aes256-cbc`, `tripledes-cbc`, etc ...The default is `tripledes-cbc` if not specified. Values are from the XML Encryption standard found at XML Encryption Syntax and Processing (http://www.w3.org/TR/2002/REC-xmlenc-core-20021210)
If an XMLDSIG template is used, the `SessionKeyAlgo` element need not be specified.

CertificateSearchType – Indicates identifier type to be used to retrieve the public encryption certificate from the configured LDAP Directory. Valid types are `File`, `DistinguishedName`, `DN`, and `URL`.

CertificateID – A unbounded string element containing the actual value of the `File`, `DN`, `URL` or `DistinguishedName` of the requested encryption certificate. This parameter will be used to retrieve the public encryption certificate through an LDAP or directory lookup.

5.6.1.4 Encrypt Response Object

A EncryptResponseType complex element is populated and returned by the EPM Service. Here are the elements it contains:

```
   <xs:element name="EncryptResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail"
type="epm:TransactionStatusDetailType" nillable="true" minOccurs="0"
```

```
maxOccurs="unbounded"/>
                <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
                <xs:element name="PostMarkedReceipt" type="epm:PostMarkedReceiptType"
nillable="true"/>
                <xs:element name="SignatureData" type="epm:QualifiedDataType"/>
                <xs:element name="SignatureInfo" type="epm:SignatureInfoType"
nillable="true"/>
                <xs:element name="X509Info" type="epm:X509InfoType" nillable="true"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

TransactionStatus – A string element representing the result of the overall transaction. ‘0’ denotes success. ‘1’ denotes that a warning was generated. Transaction failure is denoted by a specific numerical value..

TransactionStatusDetail – An element whose type is a complex element defined by TransactionStatusDetailType. (see TransactionStatus and TransactionStatusDetailType in subclause 5.4.1)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the EPM Service. Please refer to TransactionKeyType in subclause 5.4.2.

PostMarkedReceipt – A binary element containing the TSA-generated timestamptoken acting as the receipt. This receipt is attestation that the PostMark has been created. Please refer to subclause 5.4.6 entitled PostMarkedReceipt for details.

SignatureData –This element will contain the generated PKCS7 or XMLDSIG based `EnvelopedData` structure.

SignatureInfoType – An element whose type is a complex element defined by SignatureInfoType. Please refer to subclause 5.4.4 entitled SignatureInfoType for details.

X509InfoType – An element whose type is a complex element defined by X509InfoType. Please refer to subclause 5.4.5 entitled X509InfoType for details.

5.6.2 Decrypt

5.6.2.1 Decrypt Edit Rules Summary

The Decrypt operation requires a valid `EnvelopedData` structure as an input parameter. It may be either a PKCS7 EnvelopedData ASN.1 binary object or an XML Encryption compliant XML `EnvelopedData` node. This enveloped data object will be decrypted and returned in its original content format. Decrypting is the process whereby the enveloped data’s fixed symmetric key is decrypted, and subsequently used to decrypt the original message content. The Decrypt operation will decrypt any incoming `EnvelopedData` format. The `EnvelopedData` object shall be encrypted with the public key portion of the EPM Service’s private decryption key and belongs to the Postal Administration. It is most often used in Delegated Confidentiality scenarios as described in subclause 5.2.9.2 entitled Delegated Confidentiality Service. Please also refer to subclause 5.4.12 entitled EncryptResponse Option which covers the use of this option with examples of usage as they apply to the Verify, Decrypt, and RetrieveResults operations.

NOTE   In situations where the EPM is deployed centrally as a shared service, EPM implementations SHALL ensure that the authenticated requests which involve decryption of sensitive content honor and respect restricted `ParticipatingParty` lists within that Lifecycle.

5.6.2.2 DecryptOptions Request Flags

The following option flags can be set in the DecryptOptionsType complex element. These elements are of type boolean and should be set to ‘True’ or ‘False’, the default value being ‘False’. This object will be added as an element reference to the `DecryptRequestType` complex element which is referenced in the next subclause.

```
   <xs:complexType name="DecryptOptionsType">
      <xs:sequence>
         <xs:element name="EndLifecycle" type="xs:boolean"/>
         <xs:element name="ExtendLifecycle" type="xs:boolean"/>
         <xs:element name="EncryptResponse" type="xs:boolean"/>
         <xs:element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
         <xs:element name="ReturnSignatureInfo" type="xs:boolean"/>
         <xs:element name="ReturnX509Info" type="xs:boolean"/>
      </xs:sequence>
   </xs:complexType>
```

EndLifecycle – Set to ‘True’ to end the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

ExtendLifecycle – Set to ‘True’ to extend the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

EncryptResponse – This option requires that the caller sign the request by initializing the `ClaimedIdentity` request element. This option instructs the EPM to encrypt the `Data` element before returning it using the public key present in the incoming `RequesterSignature`. Please also refer to subclause 5.4.12 entitled EncryptResponse Option which covers the use of this option with examples of usage as they apply to the Verify, Decrypt, and RetrieveResults operations.

StoreNonRepudiationEvidence – Set to ‘True’ to turn on database logging of non-repudiation evidence. If set to ‘False’, non-repudiation information is not logged to the database. If set to ‘False’ a TransactionKey is still generated but only skeleton information relating to the transaction status is logged to the database. This skeleton information is logged solely in order to support subsequent retrieval operations and to allow this event to participate in Lifecycles.

ReturnSignatureInfo – Set to ‘True’ to return a detailed breakdown of the signature object, including the original content. Please refer to subclause 5.4.4 entitled SignatureInfoType for details.

ReturnX509Info – a Set to ‘True’ to return a detailed breakdown of the certificate used to perform the operation. Please refer to subclause 5.4.5 entitled X509InfoType for details.

5.6.2.3 Decrypt Request Elements

Set the following parameters or elements of the DecryptRequestType complex element before invoking the EPM Service as appropriate:

```
   <xs:element name="DecryptRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="DecryptOptions" type="epm:DecryptOptionsType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"
nillable="true"/>
            <xs:element name="ClaimedIdentity" type="epm:ClaimedIdentityType"
nillable="true"/>
            <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
            <xs:element name="ClientApplication" type="epm:ClientApplicationType"/>
            <xs:element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <xs:element name="EnvelopedData" type="epm:QualifiedDataType"/>
            <xs:element name="ContentMetadata" type="epm:ContentMetadataType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

DecryptOptions – An element whose type is a complex element defined by `DecryptOptionsType` from the previous subclause.

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key and Sequence elements should be initialized as null for Decrypt requests unless one is extending a lifecycle. Please refer to TransactionKeyType in subclause 5.4.2.

ClaimedIdentity – The RequesterSignature element of ClaimedIdentity shall be initialized when either Proof-of-Delivery or Proof-of-Possession is desired by the transacting parties, or when this operation is part of a `StartLifecycle` whose `AccessLevel` is `Signed.` See also

ClaimedIdentity in subclause 5.4.10.

OrganizationID – A string element representing the identifier of the organization requesting the service. This value must match against a list of valid Organization Identifiers that are registered with the EPM Service.

ClientApplication – See ClientApplication subclause 5.4.17

ContentIdentifier – See ContentIdentifier subclause 5.4.18

EnvelopedData – The enveloped data to be decrypted in either encapsulated ASN.1 binary CMS/PKCS7v1.5 or the XML Encryption standard. In either case, the application shall present the envelope to be decrypted as an octet-stream to the SOAP layer which will base64 encode it for transport. The EPM will determine the envelope format from the MimeType attribute. Valid values are either `text/xml`, in which case the EPM will assume that the caller is passing in an XMLENC-formatted envelope, or "application/pkcs7-signature", in which case the EPM will assume the caller is passing in ASN.1 binary CMS/PKCS7..

ContentMetaData – A string element containing custom details of the encrypted data that can be specified by the client. Example usage could be the original file name, file date, file size or file owner information.

5.6.2.4 Decrypt Response Object

A DecryptResponseType complex element is populated and returned by the EPM Service. Here are the elements it contains:

```
   <xs:element name="DecryptResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail"
type="epm:TransactionStatusDetailType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
            <xs:element name="Data" type="epm:QualifiedDataType"/>
            <xs:element name="SignatureInfo" type="epm:SignatureInfoType"
nillable="true"/>
            <xs:element name="X509Info" type="epm:X509InfoType" nillable="true"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

TransactionStatus – A string element representing the result of the overall transaction. ‘0’ denotes success. ‘1’ denotes that a warning was generated. Transaction failure is denoted by a specific numerical value..

TransactionStatusDetail – An element whose type is a complex element defined by TransactionStatusDetailType. See TransactionStatus and TransactionStatusDetailType in subclause 5.4.1

TransactionKey – An element whose type is a complex element defined by TransactionKeyType.

Locator, Key, and Sequence elements will be populated and returned by the EPM Service. Please refer to TransactionKeyType in subclause 5.4.2.

Data – A binary element containing the decrypted content. If the `EncryptResponse` option was set true, this element will contain an `EnvelopedData` object encrypted for the caller. The `MimeType` will and the context will allow the caller to discern which type it is.

SignatureInfoType – An element whose type is a complex element defined by SignatureInfoType. Please refer to subclause 5.4.4 entitled SignatureInfoType for details.

X509InfoType – An element whose type is a complex element defined by X509InfoType. Please refer to subclause 5.4.5 entitled X509InfoType for details.

5.6.3 Locate

5.6.3.1 Locate Edit Rules Summary

The Locate operation requires a valid 'CertificateSearchType' and a 'CertificateID' as input parameters. This operation will retrieve an encryption certificate based on the criteria one specifies in the `LocateRequestType` complex element. Locate is required to support public certificate access and information retrieval.

5.6.3.2 LocateOptions Request Flags

The following option flags can be set in the LocateOptionsType complex element. These elements are of type boolean and should be set to 'True' or 'False', the default value being 'False'. This object will be added as an element reference to the `LocateRequestType` complex element which is referenced in the next subclause.

```
<xs:complexType 4yLocateOptionsType">
    <xs:sequence>
        <xs:element name="EndLifecycle" type="xs:boolean"/>
        <xs:element name="ExtendLifecycle" type="xs:boolean"/>
        <xs:element name="VerifyCertificate" type="xs:boolean"/>
        <xs:element name="ReturnX509Info" type="xs:boolean"/>
    </xs:sequence>
</xs:complexType>
```

EndLifecycle – Set to 'True' to end the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

VerifyCertificate – Set to 'True' to enable certificate revocation checking. If set to 'False', the certificate revocation checking will be bypassed, but the standard certificate validation will still occur. All certificate validation results will be returned in the X509InfoType element. Please refer to subclause 5.4.5 entitled X509InfoType for details..

ReturnX509 Info – Set to 'True' to return a detailed breakdown of the certificate used to perform the operation. Please refer to subclause 5.4.5 entitled X509InfoType for details.

5.6.3.3 Locate Request Elements

Set the following parameters or elements of the LocateRequestType complex element before invoking the EPM Service as appropriate:

```
<xs:element name="LocateRequest">
```

```
      <xs:complexType>
         <xs:sequence>
            <xs:element name="LocateOptions" type="epm:LocateOptionsType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"
nillable="true"/>
            <xs:element name="ClaimedIdentity" type="epm:ClaimedIdentityType"
nillable="true"/>
            <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
            <xs:element name="ClientApplication" type="epm:ClientApplicationType"/>
            <xs:element name="CertificateSearchType" type="xs:string"/>
            <xs:element name="CertificateID" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

LocateOptions – An element whose type is a complex element defined by `LocateOptionsType` from the previous subclause.

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key and Sequence elements should be initialized as null for Locate requests unless one is extending a lifecycle. Please refer to TransactionKeyType in subclause 5.4.2.

ClaimedIdentity – The `RequesterSignature` element of `ClaimedIdentity` shall be initialized when either Proof-of-Delivery or Proof-of-Possession is desired by the transacting parties, or when this operation is part of a `StartLifecycle` whose `AccessLevel` is `Signed.` See also

ClaimedIdentity in subclause 5.4.10.

OrganizationID – A string element representing the identifier of the organization requesting the service. This value must match against a list of valid Organization Identifiers that are registered with the EPM Service.

ClientApplication – See ClientApplication subclause 5.4.17

CertificateSearchType – Indicates identifier type to be used to retrieve the public encryption certificate from the configured LDAP Directory. Valid types are `File`, `DistinguishedName`, `DN`, and `URL`.

CertificateID – A string element containing the actual value of the `File`, `DN`, `URL` or `DistinguishedName` of the requested encryption certificate. This parameter will be used to retrieve the public encryption certificate through the LDAP lookup.

5.6.3.4 Locate Response Object

A LocateResponseType complex element is populated and returned by the EPM Service. Here are the elements it contains:

```
   <xs:element name="LocateResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail"
type="epm:TransactionStatusDetailType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
            <xs:element name="PublicEncryptionCert" type="xs:string"/>
            <xs:element name="X509Info" type="epm:X509InfoType" nillable="true"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

TransactionStatus – A string element representing the result of the overall transaction. ‘0’ denotes success. ‘1’ denotes that a warning was generated. Transaction failure is denoted by a specific numerical value..

TransactionStatusDetail – An element whose type is a complex element defined by TransactionStatusDetailType. (See TransactionStatus and TransactionStatusDetailType in subclause 5.4.1)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the EPM Service. Please refer to TransactionKeyType in subclause 5.4.2.

PublicEncryptionCert – A binary element containing the actual returned encryption certificate.

X509InfoType – An element whose type is a complex element defined by X509InfoType. Please refer to subclause 5.4.5 entitled X509InfoType for details.

5.6.4 LogEvent

5.6.4.1 LogEvent Edit Rules Summary

The LogEvent is available to allow customers to log and ***system*** timestamp any content they believe is of significance to the business workflow or life cycle. This facility is also useful when the EPM is participating with other system components which are supporting other events within the lifecycle which shall be logged. An example might be a Single-Sign-On service working in conjunction with the EPM that has been delegated under the Signature Policy to be responsible and accountable for client authentication prior to EPM Service consumption. Another example of its use might be for purposes of non-repudiation of transport which is provided by another system component outside the EPM. The transport service would send a LogEvent attesting to successful delivery of a document. Working in conjunction with this transport service the EPM would be able to provide the complete non-repudiation story from origin, though submission and delivery and concluding with receipt. Yet another scenario arises when customers have extremely large payloads which would be inefficient to cryptographically process. In this scenario the client can hash to content and sign the hash. This signature Verify event can be sent up to the EPM as the first event in a Lifecycle. The second event in this Lifecycle can be the content itself sent up on a LogEvent operation.

NOTE Clients wishing to have stronger non-repudiability and authenticity involving cryptographic timestamping and PostMarking should use the PostMark operation. Alternatively the participating system could sign the required content and send the EPM a Verify and Postmark request to close the loop in a binding way.

5.6.4.2 LogEventOptions Request Flags

The following option flags can be set in the LogEventOptionsType complex element. These elements are of type boolean and should be set to ‘True’ or ‘False’, the default value being ‘False’. This object will be added as an element reference to the `LogEventRequestType` complex element which is referenced in the next subclause.

```
<xs:complexType name="LogEventOptionsType">
   <xs:sequence>
      <xs:element name="EndLifecycle" type="xs:boolean"/>
      <xs:element name="ExtendLifecycle" type="xs:boolean"/>
      <xs:element name="DecryptIncomingEnvelope" type="xs:boolean"/>
   </xs:sequence>
</xs:complexType>
```

EndLifecycle – Set to ‘True’ to end the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

DecryptIncomingEnvelope – Set to 'True' to instruct the EPM Service to decrypt the `Data` element before performing the LogEvent operation. This element needs to be included as an EnvelopedData object and must be encrypted with the public key portion of the EPM Service's private decryption key. This option is used to ensure confidential delivery of the transaction content to the EPM Service. Please also refer to subclause 5.4.12 entitled EncryptResponse Option for more details.

5.6.4.3 LogEvent Request Elements

Set the following parameters or elements of the LogEventRequestType complex element before invoking the EPM Service as appropriate:

```
   <xs:element name="LogEventRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="LogEventOptions" type="epm:LogEventOptionsType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"
nillable="true"/>
            <xs:element name="ClaimedIdentity" type="epm:ClaimedIdentityType"
nillable="true"/>
            <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
            <xs:element name="ClientApplication" type="epm:ClientApplicationType"/>
            <xs:element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <xs:element name="Data" type="epm:QualifiedDataType"/>
            <xs:element name="ContentMetadata" type="epm:ContentMetadataType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

LogEventOptions – An element whose type is a complex element defined by `LogEventOptionsType` from the previous subclause.

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key and Sequence elements should be initialized as null for LogEvent requests unless one is extending a lifecycle. Please refer to TransactionKeyType in subclause 5.4.2.

ClaimedIdentity – The `RequesterSignature` element of `ClaimedIdentity` shall be initialized when either Proof-of-Delivery or Proof-of-Possession is desired by the transacting parties, or when this operation is part of a `StartLifecycle` whose `AccessLevel` is `Signed.` See also

ClaimedIdentity in subclause 5.4.10.

OrganizationID – A string element representing the identifier of the organization requesting the service. This value must match against a list of valid Organization Identifiers that are registered with the EPM Service.

ClientApplication – See ClientApplication subclause 5.4.17

ContentIdentifier – See ContentIdentifier subclause 5.4.18

Data – A binary element initialized by the caller as a language-specific octet stream representing the data to be stored in the EPM Database.

ContentMetadata – See ContentMetadata

5.6.4.4 LogEvent Response Object

A LogEventResponseType complex element is populated and returned by the EPM Service. Here are the elements it contains:

```
   <xs:element name="LogEventResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail"
type="epm:TransactionStatusDetailType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

TransactionStatus – A string element representing the result of the overall transaction. ‘0’ denotes success. ‘1’ denotes that a warning was generated. Transaction failure is denoted by a specific numerical value..

TransactionStatusDetail – An element whose type is a complex element defined by TransactionStatusDetailType. (see TransactionStatus and TransactionStatusDetailType in subclause 5.4.1)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the EPM Service. Please refer to TransactionKeyType in subclause 5.4.2.

5.6.5 StartLifeCycle

5.6.5.1 StartLifecycle Edit Rules Summary

The StartLifeCycle operation is used to start an extended business transaction lifecycle for a given series of operations (See Lifecycle Management in subclause 5.3.3 for more information.). The operation requires either a valid participating party group (refer to ParticipatingPartyType in subclause 5.4.9 for details) unless the `AccessScope` element is set to `Global` (Please refer to subclause 5.4.11 entitled AccessScope and Scopes for further details). This will restrict who can access, participate in, and contribute to this lifecycle.

The StartLifecycle operation is an optional operation. It needs to be specified if the subscriber wishes to specify a `ParticipatingParty` list. If the subscriber does not wish to specify `ParticipatingParty` entries and is willing to allow a value of `Global` for the `AccessScope` element, and a value of `default` for the `AccessLevel` element, then an explicit `StartLifecycle` operation is not required and Lifecycles may still be created by simply initializing the `TransactionKey` to a valid key value. This is termed implicit Lifecycle support. All transactions return a `TransactionKey`. Any operation can initialize the `TransactionKey` to a value of some known previous transaction event, turn on the `ExtendLifecycle` option, and thus extend the Lifecycle.

5.6.5.2 StartLifecycleOptions Request Flags

None – A StartLifeCycleOptionsType element does not exist.

5.6.5.3 StartLifecycle Request Elements

Set the following parameters or elements of the StartLifeCycleRequestType complex element before invoking the EPM Service as appropriate:

```
   <xs:element name="StartLifecycleRequest">
      <xs:complexType>
         <xs:sequence>
```

```
            <xs:element name="ClaimedIdentity" type="epm:ClaimedIdentityType"
nillable="true"/>
            <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
            <xs:element name="ClientApplication" type="epm:ClientApplicationType"/>
            <xs:element name="ParticipatingParty" type="epm:ParticipatingPartyType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="AccessScope" type="epm:Scopes"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

ClaimedIdentity – The `RequesterSignature` element of `ClaimedIdentity` shall be initialized when either Proof-of-Delivery or Proof-of-Possession is desired by the transacting parties, or when this operation is part of a `StartLifecycle` whose `AccessLevel` is `Signed.` See also

ClaimedIdentity in subclause 5.4.10.

OrganizationID – A string element representing the identifier of the organization requesting the service. This value must match against a list of valid Organization Identifiers that are registered with the EPM Service.

ClientApplication – See ClientApplication subclause 5.4.17

ParticipatingParty – An element whose type is a complex element defined by ParticipatingPartyType. (refer to ParticipatingPartyType in subclause 5.4.9 for details)

AccessScope – An element containing a simple object defined as Scopes. Valid values are `Global`, `Organizational`, `Individual`, and `Mixed`.

5.6.5.4 StartLifecycle Response Object

A StartLifeCycleResponseType complex element is populated and returned by the EPM Service. Here are the elements it contains:

```
   <xs:element name="StartLifecycleResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail"
type="epm:TransactionStatusDetailType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

TransactionStatus – A string element representing the result of the overall transaction. '0' denotes success. '1' denotes that a warning was generated. Transaction failure is denoted by a specific numerical value..

TransactionStatusDetail – An element whose type is a complex element defined by TransactionStatusDetailType. (see TransactionStatus and TransactionStatusDetailType in subclause 5.4.1)

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key, and Sequence elements will be populated and returned by the EPM Service. Please refer to TransactionKeyType in subclause 5.4.2.

5.6.6 RetrieveSummary

5.6.6.1 RetrieveSummary Edit Rules Summary

The RetrieveSummary operation can be used for 2 main purposes. First, to access a summary list of all the events that have taken place in a given Lifecycle designated by a given TransactionKey. Second, to access all the Transaction Keys for a given OrganizationID based on selection criteria. Once the desired event has been located with this operation since it returns a list of `TransactionKey`'s that met the criteria, the client can then access the specific details of the retrieved events using the RetrieveResults operation outlined above. The `RequesterSignature` element of `ClaimedIdentity` shall be supplied.

RetrieveSummary can only be executed by individuals from the organization which created the events being retrieved. In other words only organizations can access their own data.

5.6.6.2 RetrieveSummaryOptions Request Flags

The following parameter or element is to be set in the RetrieveSummaryOptionsType complex element. These elements are of type boolean and should be set to 'True' or 'False', the default value being 'False'. This object will be added as an element reference to the `RetrieveSummaryRequestType` complex element which is referenced in the next subclause.

```
   <xs:complexType name="RetrieveSummaryOptionsType">
      <xs:sequence>
         <xs:element name="EndLifecycle" type="xs:boolean"/>
         <xs:element name="ExtendLifecycle" type="xs:boolean"/>
      </xs:sequence>
   </xs:complexType>
```

EndLifecycle – Set to 'True' to end the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

ExtendLifecycle – Set to 'True' to extend the transaction lifecycle. (See Lifecycle Management in subclause 5.3.3 for more information.)

5.6.6.3 RetrieveSummary Request Elements

Set the following parameters or elements of the RetrieveSummaryRequestType complex element before invoking the EPM Service as appropriate.

```
   <xs:element name="RetrieveSummaryRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="RetrieveSummaryOptions"
type="epm:RetrieveSummaryOptionsType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"
nillable="true"/>
            <xs:element name="ClaimedIdentity" type="epm:ClaimedIdentityType"
nillable="true"/>
            <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
            <xs:element name="ClientApplication" type="epm:ClientApplicationType"/>
            <xs:element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <xs:element name="LastUniqueSequenceId" type="xs:string"
nillable="true"/>
            <xs:element name="HashValue" type="xs:string" nillable="true"/>
            <xs:element name="StartDateTime" type="xs:string" nillable="true"/>
            <xs:element name="EndDateTime" type="xs:string" nillable="true"/>
            <xs:element name="ContentMetadata" type="epm:ContentMetadataType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="RetrieveCount" type="xs:string" nillable="true"/>
         </xs:sequence>
      </xs:complexType>
   <xs:element>
```

RetrieveSummaryOptions – An element whose type is a complex element defined by `RetrieveSummaryOptionsType` from the previous subclause.

TransactionKey – An element whose type is a complex element defined by TransactionKeyType. Locator, Key and Sequence elements should be initialized as 'null' for Verify requests unless one is extending a lifecycle. Please refer to TransactionKeyType in subclause 5.4.2.

ClaimedIdentity – The RequesterSignature element of ClaimedIdentity shall be initialized when either Proof-of-Delivery or Proof-of-Possession is desired by the transacting parties, or when this operation is part of a `StartLifecycle` whose `AccessLevel` is `Signed.` See also

ClaimedIdentity in subclause 5.4.10.

OrganizationID – A string element representing the identifier of the organization requesting the service. This value must match against a list of valid Organization Identifiers that are registered with the EPM Service.

ClientApplication – See ClientApplication subclause 5.4.17

ContentIdentifier – See ContentIdentifier subclause 5.4.18

LastUniqueSequenceId – This element allows an application to retrieve transactions which satisfy the selection criteria after a particular sequence number within the EPM's database. This is useful in situations where nightly processing takes place and the application wishes to resume where it left off.

HashValue – Allows the caller to retrieve Sign or Verify transactions with a particular hash value.

StartDateTime – Allows the caller to specify and date and time range for the transactions to be selected.

EndDateTime – Allows the caller to specify and date and time range for the transactions to be selected.

ContentMetadata – Allows the caller to match transactions on the `ContentMetadata` passed in on the original transaction.

RetrieveCount – Limits the number of transactions retrieved in the response to a particular count. Can be used in conjunction with the `LastUniqueSequenceId` above.

5.6.6.4 RetrieveSummary Response Object

A RetrieveSummaryResponseType complex element is populated and returned by the EPM Service. Here are the elements it contains:

```
   <xs:element name="RetrieveSummaryResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail" 
type="epm:TransactionStatusDetailType" nillable="true" minOccurs="0" 
maxOccurs="unbounded"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
            <xs:element name="RetrieveSummaryInfo" type="epm:RetrieveSummaryInfoType" 
minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

RetrieveSummaryInfo – An element whose type is a complex element defined by RetrieveSummaryInfoType. An occurrence of this element will exist for each transaction that matches the criteria specified in the request. The elements returned are from the selected transaction and allow the application to decide if they wish to obtain further detail by issuing a RetrieveResults operation on the retrieved key.

```
   <xs:complexType name="RetrieveSummaryInfoType">
```

```
      <xs:sequence>
         <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
         <xs:element name="UniqueSequenceId" type="xs:string" nillable="true"/>
         <xs:element name="Operation" type="xs:string"/>
         <xs:element name="OrganizationID" type="xs:string" nillable="true"/>
         <xs:element name="ClientApplication" type="epm:ClientApplicationType"
nillable="true"/>
         <xs:element name="ContentIdentifier" type="xs:string" nillable="true"/>
         <xs:element name="TransactionStatus" type="xs:string"/>
         <xs:element name="TimeStampValue" type="xs:string" nillable="true"/>
         <xs:element name="ContentHash" type="xs:string" nillable="true"/>
         <xs:element name="SigningTime" type="xs:string" nillable="true"/>
         <xs:element name="X509Subject" type="xs:string" nillable="true"/>
         <xs:element name="X509Issuer" type="xs:string" nillable="true"/>
         <xs:element name="X509Serial" type="xs:string" nillable="true"/>
      </xs:sequence>
   </xs:complexType>
```

5.6.7 RetrievePostalAttributes

5.6.7.1 RetrievePostalAttributes Edit Rules Summary

The RetrievePostalAttributes operation is used to access a list of localization attributes that are specific to a country or region's EPM service provider. This operation is used to retrieve a list of country-specific attributes by category, where each attribute is maintained as a Name/Value pair keyed by Locator and maintained in a “Yellow Pages” like directory. These attributes are used to customize the visibility of a country-issued `PostMarkedReceipt` when that receipt is viewed outside the country of receipt origin. Since the `PostMarkedReceipt` contains a `Locator` element, this element can be used to access receipt rendering detail specific to the country of receipt origin.

Examples of attributes which can be specified by a post for PostMarkedReceipt tailoring include:

- Country-specific logos
- Language-specific receipt text
- Labels and captions on User Interface (UI) controls
- Customized dialog actions by country

The attributes can be retrieved once and stored locally by any EPM-compliant desktop application. These attributes are organized by category by ResourceID. Each ResourceID can represent a series of Templates expressed in XML which are used for post-specific tailoring of any visibility aspect of the EPM’s desktop appearance. Please consult the EPM Common System Development Kit for more details on the integration and use of the EPM’s User Interface Toolkit and SDK.

```
   <xs:element name="RetrievePostalAttributesRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Locator" type="epm:LocatorType"/>
            <xs:element name="LanguageCode" type="xs:string"/>
            <xs:element name="ClientApplication" type="epm:ClientApplicationType"
nillable="true"/>
            <xs:element name="AttributeCategory" type="xs:string" nillable="true"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

This element is used to define localization attributes that are specific to a country or region's EPM service provider.

```
   <xs:element name="RetrievePostalAttributesResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="TransactionStatus" type="xs:string"/>
            <xs:element name="TransactionStatusDetail"
type="epm:TransactionStatusDetailType"/>
            <xs:element name="TransactionKey" type="epm:TransactionKeyType"/>
            <xs:element name="PostalAttribute" type="epm:PostalAttributeType"
nillable="true" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   <xs:complexType name="PostalAttributeType">
      <xs:sequence>
         <xs:element name="AttributeName" type="xs:string"/>
         <xs:element name="LastModifiedDateTime" type="xs:string"/>
         <!-- YYYYMMDDHHMMSSTTT -->
         <xs:element name="AttributeTextValue" type="xs:string" nillable="true"/>
         <xs:element name="AttributeBinaryValue" type="xs:base64Binary"
nillable="true"/>
      </xs:sequence>
   </xs:complexType>
```

ANNEX A (INFORMATIVE) EUROPEAN AND INTERNATIONAL STANDARDS INTER-RELATIONSHIPS AND EVOLUTION

This Annex discusses the role and influence of existing signature standards which exist in the same domain and scope as the EPM. Their influence and role in shaping the EPM and its evolution is also covered.

The European Directive on a community framework for electronic signatures defines an electronic signature as: “data in electronic form which is attached to or logically associated with other electronic data and which serves as a method of authentication”. An electronic signature as defined in TS 101 733 “Electronic Signature Formats” is a form of advanced electronic signature as defined in the European Directive.

ETSI TS 101 733 defines formats for electronic signatures that are compliant with the European Directive. Currently, the ETSI standard uses Abstract Syntax Notation 1 to define the structure of the electronic signature. The structure of the electronic signature defined in TS 101 733 is based on the structure defined in RFC 2630: “Cryptographic Message Syntax”. TS 101 733 satisfies the requirements made by the European Directive by defining new ASN.1 structures that can be added as parts of the fields “signedAttrs” and “unsignedAttrs”.

As a consequence of the growing importance of the use of XML on Internet, a standard for XML based digital signatures is currently being produced within W3C and IETF Working Group “XML-Signature Core Syntax and Processing”. ETSI is in the process of producing a technical specification ETSI TS 101 903: “XML Advanced Electronic Signatures (XAdES)” that defines a XML format for electronic signatures that are compliant with the European Directive, as TS 101 733 does for ASN.1 syntax. An electronic signature produced in accordance with that document provides evidence that can be processed to get confidence that some commitment has been explicitly endorsed under a Signature policy, at a given time, by a signatory under an identifier, e.g., a name or a pseudonym, and optionally a role.

TS 101 733 also deals with the signature policy issue. Although the present document does not mandate any form of signature policy specification, it specifies an ASN.1 based syntax that may be used to define a structured signature policy in a way that machines can read and process. The present report deals with the specification of new XML elements able to contain the signature policy information specified in TS 101 733.

There exists a very close evolution of both RFC 3126 and TS 101 733; as well as RFC 3275 and TS 101 933, the latter two covering XML Digital Signature Syntax and Processing. It is the EPM Standardisation Team’s hope that by staying abreast of these developments as they mature, the standard EPM itself will remain consistent with the industry direction in this area. It should also be noted that the EPM WSDL interface specification is a layer above the standards described in this subclause. Just as the EPM makes uses of time stamping standards covered by RFC 3161, it also makes use of other digital signature formatting standards such as CMS and XMLDSIG. It aims to bring these standards together into a more non-repudiation centric standard, which the other standards on their own do not accomplish.

ANNEX B (INFORMATIVE) EXAMPLES

This subclause is referred to throughout the text and contains specific examples illustrating the various constructs used within the interface.

PostMarked receipts are normally returned to the application as standalone XML structures, whether they are of type CMS/PKCS7 or XMLSig. Upon request however `PostMarkedReceipt`’s can be embedded in the incoming signed document. This is true for both the Sign protocol as well as the Verify protocol. The first example below is a standalone `PostMarkedReceipt`, and the second example is one that is embedded into the signed document.

EXAMPLE 1 – STANDALONE POSTMARKEDRECEIPT OVER A VERIFIED SIGNATURE

This is an example of a **standalone** `PostMarkedReceipt` element which would be returned after a successful Verify operation. The `PostMarkedReceipt` is formatted as standalone because the user has specified a value of `standalone` in the `Location` sub-element of the `IssuePostMarkedReceipt` option on the Verify request. It would be similarly formatted if requested on a Sign operation as well. It is essentially a conventional XMLDSig enveloping signature over the `<SignatureValue>`’s of the target signature(s) being PostMarked. It contains three (3) <Reference> elements pointing to each of the following:

- a standard `<dss:TstInfo>` as per [DSSCore]
- an `<epm:PostMarkedReceipt>` element from the [EPM] schema
- the `<SignatureValue>` element of the target signature being PostMarked

Selected element contents have been deliberately truncated for brevity and clarity.

```
<?xml version="1.0" encoding="UTF-8"?>
<dsig:Signature Id="PostMarkedReceipt_001" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:SignedInfo>
      <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <dsig:Reference URI="#TstInfo040327174718Z">
         <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <dsig:DigestValue>jWkUFR6epvkrtaxTiQ33DiWy+l8=</dsig:DigestValue>
      </dsig:Reference>
      <dsig:Reference URI="#Receipt040327174718Z">
         <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <dsig:DigestValue>9JWKdLh/8Cs9Slu2QmZixOJl+x0=</dsig:DigestValue>
      </dsig:Reference>
      <dsig:Reference URI="#PostMarkedSignature">
         <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <dsig:DigestValue>MOBYPfrllMBcJz6yojbhrwH9KP4=</dsig:DigestValue>
      </dsig:Reference>
   </dsig:SignedInfo>
   <dsig:SignatureValue>qnBvJoSgo4OoiYYaE3AwbL5/EDq7BhTT6 ... Qw11HK+zxy66I=</dsig:SignatureValue>
   <dsig:KeyInfo>
      <dsig:KeyName>C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, E= … </dsig:KeyName>
      <dsig:X509Data>
      <X509Certificate xmlns="http://www.w3.org/2000/09/xmldsig#">MIIEUDC … EwZOBg==</X509Certificate>
<X509SubjectName xmlns="http:// … xmldsig#"> C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, E=… </X509SubjectName>
      <X509IssuerSerial xmlns="http://www.w3.org/2000/09/xmldsig#">
         <X509IssuerName>C=CA, O=CPC, OU=EPM Service, CN=Electronic PostMark CA, E=… </X509IssuerName>
         <X509SerialNumber>25</X509SerialNumber>
      </X509IssuerSerial>
      </dsig:X509Data>
   </dsig:KeyInfo>
   <dsig:Object xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dss:TstInfo xmlns:dss="http://www.docs.oasis-open.org/dss/2004/06/oasis-dss-1.0-core-schema-wd-30.xsd"
                  Id="TstInfo040327174718Z">
         <SerialNumber>1847365279</SerialNumber>
         <CreationTime>2004-03-27T17:47:18.750</CreationTime>
         <Policy/>
         <ErrorBound/>
         <Ordered/>
         <TSA>C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, E= … </TSA>
      </dss:TstInfo>
   </dsig:Object>
   <dsig:Object xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <epm:PostMarkedReceipt xmlns:epm="http://www.upu.int/EPMService" Id="Receipt040327174718Z">
         <Receipt>
            <TransactionKey>
               <Locator>
                  <CountryCode>CA</CountryCode>
                  <Version>114</Version>
                  <ServiceProvider>ePost Corporation</ServiceProvider>
                  <Environment xsi:nil="true"/>
               </Locator>
               <Key>1234567890</Key>
               <Sequence>1</Sequence>
            </TransactionKey>
            <Requester>CN=Joe Public, O=VeriSign Class 1 Certificate, C=CA, E=joe.public@rogers.com</Requester>
            <Operation>Verify</Operation>
            <RevocationStatusQualifier>CRL Checked</RevocationStatusQualifier>
            <TimeStampToken MimeType="application/pkcs7-signature"></TimeStampToken>
            <Metadata>
               <Name>TSAX509SubjectName</Name>
               <Value>C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, … </Value>
            </Metadata>
            <Metadata>
               <Name>TimeStampValue</Name>
               <Value>040327174718Z</Value>
            </Metadata>
         </Receipt>
      </epm:PostMarkedReceipt>
   </dsig:Object>
   <dsig:Object xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <epm:PostMarkedSignatureValue xmlns:epm="http://www.upu.int/EPMService"
           Id="PostMarkedSignature">Vif2Y7aShziFOCy0sDUOR9XVnVCy8LW9hY ... 2RsmQETPmsM=</epm:PostMarkedSignatureValue>
   </dsig:Object>
</dsig:Signature>
```

EXAMPLE 2 – STANDALONE <POSTMARKEDRECEIPT> OVER DATA WHEN USING POSTMARK OPERATION

Similar to Example 1 above, when the `standalone` `<PostMarkedReceipt>`'s signature scope simply covers data, as when used in a PostMark operation, then the 3rd `<Reference>` will be to an `<Object>` containing the **hash** of the data to be PostMarked with base64 encoding specified.

```
<?xml version="1.0" encoding="UTF-8"?>
<dsig:Signature Id="PostMarkedReceipt_001" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:SignedInfo>
      <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <dsig:Reference URI="#TstInfo040327174718Z">
         <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <dsig:DigestValue>jWkUFR6epvkrtaxTiQ33DiWy+l8=</dsig:DigestValue>
      </dsig:Reference>
      <dsig:Reference URI="#Receipt040327174718Z">
         <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <dsig:DigestValue>9JWKdLh/8Cs9Slu2QmZixOJl+x0=</dsig:DigestValue>
      </dsig:Reference>
      <dsig:Reference URI="#PostMarkedDataContent">
         <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <dsig:DigestValue>MOBYPfrllMBcJz6yojbhrwH9KP4=</dsig:DigestValue>
      </dsig:Reference>
   </dsig:SignedInfo>
   <dsig:SignatureValue>qnBvJoSgo4OoiYYaE3AwbL5/EDq7BhTT6 ... Qw11HK+zxy66I=</dsig:SignatureValue>
   <dsig:KeyInfo>
      <dsig:KeyName>C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, E= … </dsig:KeyName>
      <dsig:X509Data>
      <X509Certificate xmlns="http://www.w3.org/2000/09/xmldsig#">MIIEUDC … EwZOBg==</X509Certificate>
<X509SubjectName xmlns="http:// … xmldsig#"> C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, E=… </X509SubjectName>
      <X509IssuerSerial xmlns="http://www.w3.org/2000/09/xmldsig#">
         <X509IssuerName>C=CA, O=CPC, OU=EPM Service, CN=Electronic PostMark CA, E=… </X509IssuerName>
         <X509SerialNumber>25</X509SerialNumber>
      </X509IssuerSerial>
      </dsig:X509Data>
   </dsig:KeyInfo>
   <dsig:Object xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dss:TstInfo xmlns:dss="http://www.docs.oasis-open.org/dss/2004/06/oasis-dss-1.0-core-schema-wd-30.xsd"
                   Id="TstInfo040327174718Z">
```

```
            <SerialNumber>1847365279</SerialNumber>
            <CreationTime>2004-03-27T17:47:18.750</CreationTime>
            <Policy/>
            <ErrorBound/>
            <Ordered/>
            <TSA>C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, E= … </TSA>
         </dss:TstInfo>
      </dsig:Object>
      <dsig:Object xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
         <epm:PostMarkedReceipt xmlns:epm="http://www.upu.int/EPMService" Id="Receipt040327174718Z">
            <Receipt>
               <TransactionKey>
                  <Locator>
                     <CountryCode>CA</CountryCode>
                     <Version>114</Version>
                     <ServiceProvider>ePost Corporation</ServiceProvider>
                     <Environment xsi:nil="true"/>
                  </Locator>
                  <Key>1234567890</Key>
                  <Sequence>1</Sequence>
               </TransactionKey>
               <Requester>CN=Joe Public, O=VeriSign Class 1 Certificate, C=CA, E=joe.public@rogers.com</Requester>
               <Operation>PostMark</Operation>
               <RevocationStatusQualifier>Not Applicable</RevocationStatusQualifier>
               <TimeStampToken MimeType="application/pkcs7-signature"></TimeStampToken>
               <Metadata>
                  <Name>TSAX509SubjectName</Name>
                  <Value>C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, … </Value>
               </Metadata>
               <Metadata>
                  <Name>TimeStampValue</Name>
                  <Value>040327174718Z</Value>
               </Metadata>
            </Receipt>
         </epm:PostMarkedReceipt>
      </dsig:Object>
      <dsig:Object xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
         <PostMarkedData Id="PostMarkedDataContent"
            Encoding="http://www.w3.org/2000/09/xmldsig#base64">RGF0YSBgdG8gcQ...gVGkgMTUgMTI6MTA=</PostMarkedData>
      </dsig:Object>

   </dsig:Signature>
```

EXAMPLE 3 - EMBEDDED <POSTMARKEDRECEIPT> OVER A VERIFIED SIGNATURE

This is an example of an **embedded** `<PostMarkedReceipt>` returned after a successful Verify operation when the

`<IssuePostMarkedReceipt>` option element specifies `embedded` as the value of the `Location` sub-element. It is a conventional XMLSig enveloping signature over the `<SignatureValue>` of the target signature(s) being PostMarked. It contains three (3) `<Reference>` elements pointing to each of the following:

- a standard `<dss:TstInfo>` as per [DSSCore]
- an `<epm:PostMarkedReceipt>` element from the [EPM] schema
- the `<SignatureValue>` element of the target signature(s) being PostMarked

Note that depending on the value of the optional NodeName element specified within the `<IssuePostMarkedReceipt>` of the request, the PostMarkedReceipt can potentially cover all `<SignatureValue>`'s in the signed document when the document contains multiple signatures.

Selected element contents have been deliberately truncated for brevity and clarity.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Document [
<!ATTLIST Object Id ID #IMPLIED>
]>
<Document>
<!-- Beginning of PostMarkedReceipt signature -->
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dsig:SignedInfo>
         <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <dsig:Reference URI="#TstInfo040327174718Z">
            <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <dsig:DigestValue>3Lk/6TE71dqeXZFUJ9qqaPInm24=</dsig:DigestValue>
         </dsig:Reference>
         <dsig:Reference URI="#Receipt040327174718Z">
            <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <dsig:DigestValue>430zTvcoa9r8Rpr5DiVZf7IPvl8=</dsig:DigestValue>
         </dsig:Reference>
         <dsig:Reference URI="">
            <dsig:Transforms>
               <dsig:Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
                  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                                  xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" version="1.0">
                     <xsl:template match="/">
                        <xsl:copy>
                           <xsl:copy-of select="//dsig:Signature[position()!=1]/dsig:SignatureValue"/>
                        </xsl:copy>
                     </xsl:template>
                  </xsl:stylesheet>
               </dsig:Transform>
            </dsig:Transforms>
            <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <dsig:DigestValue>LRAX6mCfAq8hprb8UMU1H35PTYw=</dsig:DigestValue>
      </dsig:Reference>
   </dsig:SignedInfo>
   <dsig:SignatureValue>qnBvJoSgo4OoiYYaE3AwbL5/EDq7BhTT6 ... Qw11HK+zxy66I=</dsig:SignatureValue>
   <dsig:KeyInfo>
      <dsig:KeyName>C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, E= … </dsig:KeyName>
      <dsig:X509Data>
      <X509Certificate xmlns="http://www.w3.org/2000/09/xmldsig#">MIIEUDC … EwZOBg==</X509Certificate>
<X509SubjectName xmlns="http:// … xmldsig#"> C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, E=… </X509SubjectName>
      <X509IssuerSerial xmlns="http://www.w3.org/2000/09/xmldsig#">
         <X509IssuerName>C=CA, O=CPC, OU=EPM Service, CN=Electronic PostMark CA, E=… </X509IssuerName>
         <X509SerialNumber>25</X509SerialNumber>
      </X509IssuerSerial>
      </dsig:X509Data>
   </dsig:KeyInfo>
   <dsig:Object xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dss:TstInfo xmlns:dss="http://www.docs.oasis-open.org/dss/2004/06/oasis-dss-1.0-core-schema-wd-25.xsd"
                  Id="TstInfo040327174718Z">
         <SerialNumber>1847365279</SerialNumber>
         <CreationTime>2004-03-27T17:47:18.750</CreationTime>
         <Policy/>
         <ErrorBound/>
         <Ordered/>
         <TSAX509SubjectName>C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, … </TSAX509SubjectName>
      </dss:TstInfo>
   </dsig:Object>
   <dsig:Object xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <epm:PostMarkedReceipt xmlns:epm="http://www.upu.int/EPMService" Id="Receipt040327174718Z">
         <Receipt>
            <TransactionKey>
               <Locator>
                  <CountryCode>CA</CountryCode>
                  <Version>114</Version>
                  <ServiceProvider>ePost Corporation</ServiceProvider>
                  <Environment xsi:nil="true"/>
               </Locator>
               <Key>1234567890</Key>
               <Sequence>1</Sequence>
            </TransactionKey>
            <Requester>CN=Joe Public, O=VeriSign Class 1 Certificate, C=CA, E=joe.public@rogers.com</Requester>
            <Operation>Verify</Operation>
            <RevocationStatusQualifier>CRL Checked</RevocationStatusQualifier>
            <TimeStampToken MimeType="application/pkcs7-signature"></TimeStampToken>
            <Metadata>
               <Name>TSAX509SubjectName</Name>
               <Value>C=CA, O=CPC, OU=EPM Service, CN=EPM Signature, … </Value>
            </Metadata>
            <Metadata>
               <Name>TimeStampValue</Name>
```

```
                  <Value>040327174718Z</Value>
               </Metadata>
            </Receipt>
         </epm:PostMarkedReceipt>
      </dsig:Object>
   </dsig:Signature>
<!-- End of PostMarkedReceipt signature -->
<!-- Beginning of signed document being PostMarked -->
   <Object Id="DetachedDataBeingSigned">
      <PersonalData>
         <Name>Ed Smith</Name>
         <StreetAddress>1234 Mockingbird Lane</StreetAddress>
         <City>Yellowknife</City>
         <PostalCode>W1C6J3</PostalCode>
         <SocialInsuranceNumber>123456789</SIN>
      </PersonalData>
   </Object>
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" Id="TargetSignature">
      <dsig:SignedInfo>
         <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <dsig:Reference URI="#DetachedDataBeingSigned">
            <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <dsig:DigestValue>Po3vwPXh8kdpRUAzMGjzluao65I=</dsig:DigestValue>
         </dsig:Reference>
      </dsig:SignedInfo>
      <dsig:SignatureValue>KyKUMJKW ... Yi7swX0FjLkDDZNs=</dsig:SignatureValue>
      <dsig:KeyInfo>
         <dsig:KeyName>C=CA, O=Acme Corp, CN=Joe Public, E= … </dsig:KeyName>
         <dsig:X509Data>
         <X509Certificate xmlns="http://www.w3.org/2000/09/xmldsig#">MIIE … EwZOBg==</X509Certificate>
         <X509SubjectName> C=CA, O=Acme Corp, CN=Joe Public, E= … </X509SubjectName>
         <X509IssuerSerial>
            <X509IssuerName>C=CA, O=Partner CA, O=For Test Use Only, CN=Partner CA, E= … </X509IssuerName>
            <X509SerialNumber>25</X509SerialNumber>
         </X509IssuerSerial>
         </dsig:X509Data>
      </dsig:KeyInfo>
   </dsig:Signature>
<!-- End of signed document being PostMarked -->
</Document>
```

EXAMPLE 4 - REQUESTERSIGNATURE OVER TRANSACTIONKEY FOR ANY OPERATION IN PROTECTED LIFECYCLE

RequesterSignature example for `<SignatureType>` XMLDSIG when `<AccessLevel>` is `Signed` in the ParticipatingParty element of the StartLifecycle request:

```
<RequesterSignature>
   <TransactionKey xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <Locator>
         <CountryCode>CA</CountryCode>
         <Version>114</Version>
         <ServiceProvider>ePost Corporation</ServiceProvider>
         <Environment xsi:nil="true"/>
      </Locator>
      <Key>041019-133230-59841915</Key>
      <Sequence>1</Sequence>
   </TransactionKey>
   <OrganizationID>Acme Corporation</OrganizationID>
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dsig:SignedInfo>
         <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <dsig:Reference URI="">
               <dsig:Transforms>
                  <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
               </dsig:Transforms>
               <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
               <dsig:DigestValue>oOeneP9rCboxw53TX49N+B8Xnlc=</dsig:DigestValue>
            </dsig:Reference>
      </dsig:SignedInfo>
      <dsig:SignatureValue>M1jDjGp.....Z5n=</dsig:SignatureValue>
      <dsig:KeyInfo>
         <dsig:KeyName>C=CA, O=CPC, CN=....., E=.....</dsig:KeyName>
         <dsig:X509Data>
            <X509Certificate>MIIEU.....EwZOBg==</X509Certificate>
            <X509SubjectName> C=CA, O=CPC, CN=....., E=.....</X509SubjectName>
            <X509IssuerSerial xmlns="http://www.w3.org/2000/09/xmldsig#">
               <X509IssuerName>C=CA, S=Ontario, L=Ottawa, O=CPC, .....</X509IssuerName>
               <X509SerialNumber>25</X509SerialNumber>
            </X509IssuerSerial>
         </dsig:X509Data>
      </dsig:KeyInfo>
   </dsig:Signature>
</RequesterSignature>
```

EXAMPLE 5 – REQUESTERSIGNATURE OVER ORIGINALCONTENT WHEN USED IN A CHECKINTEGRITY OPERATION

This is an example of a Proof-of-Delivery scenario. The `<RequesterSignature>` element is initialized as follows for `<SignatureType>` XMLDSIG over `<OriginalContent>` with `MimeType` attribute of `text/plain`:

```
<RequesterSignature>
   <HashOfOriginalContent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
MimeType="test/plain">O5v7d3N6LDu+L.....Q3/rIveJ1rlTPo=</HashOfOriginalContent>
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dsig:SignedInfo>
         <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <dsig:Reference URI="">
               <dsig:Transforms>
                  <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
               </dsig:Transforms>
               <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
               <dsig:DigestValue>oOeneP9rCboxw53TX49N+B8Xnlc=</dsig:DigestValue>
            </dsig:Reference>
      </dsig:SignedInfo>
      <dsig:SignatureValue>M1jDjGp.....Z5n=</dsig:SignatureValue>
      <dsig:KeyInfo>
         <dsig:KeyName>C=CA, S=Ontario, L=Ottawa, O=CPC, CN=....., E=.....</dsig:KeyName>
         <dsig:X509Data>
            <X509Certificate xmlns="http://www.w3.org/2000/09/xmldsig#">MIIEU.....EwZOBg==</X509Certificate>
            <X509SubjectName xmlns="http://www.w3.org/2000/09/xmldsig#">C=CA, O=CPC, .....</X509SubjectName>
            <X509IssuerSerial xmlns="http://www.w3.org/2000/09/xmldsig#">
               <X509IssuerName>C=CA, S=Ontario, L=Ottawa, O=CPC, .....</X509IssuerName>
               <X509SerialNumber>25</X509SerialNumber>
            </X509IssuerSerial>
         </dsig:X509Data>
      </dsig:KeyInfo>
   </dsig:Signature>
</RequesterSignature>
```

ANNEX C (INFORMATIVE) EPM XML SCHEMA FILE V1.15

This subclause contains the full expanded content of the EPM Interface Schema XML .xsd file.

```
<schema targetNamespace="http://www.upu.int/EPMService/schemas" xmlns:epm="http://www.upu.int/EPMService/schemas"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-
core-schema.xsd"/>
   <!-- EPMService_UPU_V1.15.xsd Version 1.15 Last Updated August 15, 2005
-->
   <complexType name="TransactionStatusDetailType">
      <sequence>
         <element name="ErrorNumber" type="xs:string"/>
         <element name="ErrorMessage" type="xs:string" nillable="true"/>
      </sequence>
   </complexType>
   <complexType name="TransactionKeyType">
      <sequence>
         <element name="Locator" type="epm:LocatorType"/>
         <element name="Key" type="xs:string"/>
         <element name="Sequence" type="xs:positiveInteger" nillable="true"/>
      </sequence>
   </complexType>
   <complexType name="LocatorType">
      <sequence>
         <element name="CountryCode" type="xs:string"/>
         <element name="Version" type="xs:string"/>
         <element name="ServiceProvider" type="xs:string" nillable="true"/>
         <element name="Environment" type="xs:string" nillable="true"/>
      </sequence>
   </complexType>
   <complexType name="ClaimedIdentityType">
      <sequence>
         <element name="Name" type="epm:NameIdentifierType"/>
         <element name="SupportingInfo" type="epm:SupportingInfoType"/>
      </sequence>
   </complexType>
   <complexType name="SupportingInfoType">
      <sequence>
         <element name="BasicAuth" type="epm:BasicAuthType" nillable="true"/>
         <element name="RequesterSignature" type="epm:QualifiedDataType" nillable="true"/>
         <element name="AlternateIdentity" type="epm:AlternateIdentityType" nillable="true"/>
      </sequence>
   </complexType>
   <complexType name="AlternateIdentityType" abstract="true">
```

```
      <sequence>
         <element name="IdentityToken" type="xs:anyType"/>
      </sequence>
   </complexType>
   <complexType name="YourFavoriteIdentityTokenType">
      <complexContent>
         <extension base="epm:AlternateIdentityType">
            <sequence>
               <element name="FirstElement" type="xs:string"/>
               <element name="SecondElement" type="xs:base64Binary"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="SignaturePolicyIdentifierType" abstract="true">
      <sequence>
         <element name="SignaturePolicyIdentifier" type="xs:anyType"/>
      </sequence>
   </complexType>
   <complexType name="SomeSignaturePolicyIdentifierType">
      <complexContent>
         <extension base="epm:SignaturePolicyIdentifierType">
            <sequence>
               <element name="SigPolicyID" type="xs:anyURI"/>
               <element name="SigPolicyURL" type="xs:string"/>
               <element name="SigPolicyHashAlgo" type="xs:anyURI"/>
               <element name="SigPolicyHashValue" type="xs:string"/>
               <element name="SigPolicyUserNotice" type="xs:string" nillable="true"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="GenericValidationDataType" abstract="true">
      <sequence>
         <element name="GenericValidationData" type="xs:anyType"/>
      </sequence>
   </complexType>
   <complexType name="X509ValidationDataType">
      <complexContent>
         <extension base="epm:GenericValidationDataType">
            <sequence>
               <element name="X509ValidationData" type="epm:QualifiedDataType"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="BasicAuthType">
      <sequence>
         <element name="UserID" type="xs:string"/>
      <element name="Password" type="xs:string" nillable="true"/>
    </sequence>
  </complexType>
  <complexType name="NameIdentifierType">
    <simpleContent>
      <extension base="xs:string">
        <attribute name="NameQualifier" type="xs:string" use="optional"/>
        <attribute name="Format" type="xs:anyURI" use="optional"/>
      </extension>
    </simpleContent>
  </complexType>
  <complexType name="QualifiedDataType">
    <simpleContent>
      <extension base="xs:base64Binary">
        <attribute name="MimeType" type="xs:string" use="optional"/>
      </extension>
    </simpleContent>
  </complexType>
  <complexType name="OriginalContentType">
    <simpleContent>
      <extension base="xs:base64Binary">
        <attribute name="MimeType" type="xs:string"/>
      </extension>
    </simpleContent>
  </complexType>
  <complexType name="ContentMetadataType">
    <sequence>
      <element name="Name" type="xs:string"/>
      <element name="Value" type="xs:string"/>
    </sequence>
  </complexType>
  <complexType name="SignatureInfoType">
    <sequence>
      <element name="SignedContent" type="epm:QualifiedDataType" nillable="true"/>
      <element name="ContentHash" type="xs:string" nillable="true"/>
      <element name="ContentHashAlgo" type="xs:string" nillable="true"/>
      <element name="ContentEncryptAlgo" type="xs:string" nillable="true"/>
      <element name="SigningTime" type="xs:string" nillable="true"/>
      <element name="PKCS1" type="epm:QualifiedDataType" nillable="true"/>
    </sequence>
  </complexType>
  <complexType name="X509InfoType">
    <sequence>
      <element name="X509Subject" type="xs:string"/>
      <element name="X509Issuer" type="xs:string" nillable="true"/>
      <element name="X509Serial" type="xs:string" nillable="true"/>
      <element name="X509StatusSource" type="xs:string"/>
      <element name="X509ValidFrom" type="xs:string"/>
      <element name="X509ValidTo" type="xs:string"/>
      <element name="X509Certificate" type="xs:string" nillable="true"/>
      <element name="X509RevocationReason" type="xs:string" nillable="true"/>
      <element name="X509RevocationReasonString" type="xs:string" nillable="true"/>
      <element name="X509RevocationTime" type="xs:string" nillable="true"/>
      <element name="X509ValidationData" type="epm:X509ValidationDataType" nillable="true"/>
   </sequence>
</complexType>
<complexType name="ParticipatingPartyType">
   <sequence>
      <element name="PartyName" type="epm:PartyNameType"/>
      <element name="AccessLevel" type="epm:ValidAccessLevel"/>
      <element name="NotifyEvents" type="epm:ValidOperation" nillable="true" minOccurs="0" maxOccurs="unbounded"/>
      <element name="ContactID" type="xs:string" nillable="true"/>
   </sequence>
</complexType>
<complexType name="PartyNameType">
   <simpleContent>
      <extension base="xs:string">
         <attribute name="ScopeQualifier" type="epm:Scopes" use="optional"/>
      </extension>
   </simpleContent>
</complexType>
<simpleType name="Scopes">
   <restriction base="xs:string">
      <enumeration value="Global"/>
      <enumeration value="Organizational"/>
      <enumeration value="Individual"/>
      <enumeration value="Mixed"/>
   </restriction>
</simpleType>
<simpleType name="ValidOperation">
   <restriction base="xs:string">
      <enumeration value="Verify"/>
      <enumeration value="PostMark"/>
      <enumeration value="CheckIntegrity"/>
      <enumeration value="RetrieveResults"/>
      <enumeration value="Sign"/>
      <enumeration value="StartLifecycle"/>
      <enumeration value="LogEvent"/>
      <enumeration value="Encrypt"/>
      <enumeration value="Decrypt"/>
      <enumeration value="Locate"/>
      <enumeration value="RetrieveSummary"/>
      <enumeration value="RetrievePostalAttributes"/>
   </restriction>
</simpleType>
<simpleType name="ValidOption">
   <restriction base="xs:string">
```

```
      <enumeration value="EndLifecycle"/>
      <enumeration value="ExtendLifecycle"/>
      <enumeration value="VerifyCertificate"/>
      <enumeration value="DecryptIncomingEnvelope"/>
      <enumeration value="EncryptResponse"/>
      <enumeration value="StoreNonRepudiationEvidence"/>
      <enumeration value="IssuePostMarkedReceipt"/>
      <enumeration value="ReturnTimeStampAudit"/>
      <enumeration value="ReturnSignatureInfo"/>
      <enumeration value="ReturnX509Info"/>
   </restriction>
</simpleType>
<simpleType name="ValidLocation">
   <restriction base="xs:string">
      <enumeration value="standalone"/>
      <enumeration value="embedded"/>
   </restriction>
</simpleType>
<simpleType name="ValidQualifier">
   <restriction base="xs:string">
      <enumeration value="Checked"/>
      <enumeration value="Not Checked"/>
      <enumeration value="Not Applicable"/>
   </restriction>
</simpleType>
<simpleType name="ValidAccessLevel">
   <restriction base="xs:string">
      <enumeration value="Default"/>
      <enumeration value="Signed"/>
   </restriction>
</simpleType>
<simpleType name="ValidSignatureType">
   <restriction base="xs:string">
      <enumeration value="PKCS7"/>
      <enumeration value="PKCS7-detached"/>
      <enumeration value="XMLDSIG"/>
      <enumeration value="XMLDSIG-enveloping"/>
      <enumeration value="XMLDSIG-detached"/>
      <enumeration value="XMLDSIG-template"/>
   </restriction>
</simpleType>
<simpleType name="ValidCertificateSearchType">
   <restriction base="xs:string">
      <enumeration value="Distinguished Name"/>
      <enumeration value="DN"/>
      <enumeration value="File"/>
      <enumeration value="URL"/>
   </restriction>
   </simpleType>
   <simpleType name="ValidImageSize">
      <restriction base="xs:string">
         <enumeration value="Small"/>
         <enumeration value="Medium"/>
         <enumeration value="Large"/>
      </restriction>
   </simpleType>
   <element name="PostMarkedReceipt">
      <complexType>
         <sequence>
            <choice>
               <element name="PKCS7SignedReceipt" type="epm:PKCS7SignedReceiptType"/>
               <element ref="ds:Signature"/>
            </choice>
         </sequence>
      </complexType>
   </element>
   <complexType name="PKCS7SignedReceiptType">
      <sequence>
         <element name="Receipt" type="epm:ReceiptType"/>
         <element name="ReceiptSignature" type="epm:QualifiedDataType"/>
      </sequence>
   </complexType>
   <complexType name="ReceiptType">
      <sequence>
         <element name="TransactionKey" type="epm:TransactionKeyType"/>
         <element name="Requester" type="xs:string" nillable="true"/>
         <element name="Operation" type="epm:ValidOperation"/>
         <element name="TSAX509SubjectName" type="xs:string"/>
         <element name="TimeStampValue" type="xs:string"/>
         <element name="RevocationStatusQualifier" type="epm:ValidQualifier"/>
         <element name="TimeStampToken" type="epm:QualifiedDataType" nillable="true" minOccurs="0" maxOccurs="1"/>
         <element name="MessageImprint" type="xs:base64Binary" nillable="true"/>
         <element name="PostMarkImage" type="epm:QualifiedDataType" nillable="true"/>
         <element name="ReceiptMetadata" type="epm:ReceiptMetadataType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
      </sequence>
   </complexType>
   <complexType name="ReceiptMetadataType">
      <sequence>
         <element name="Name" type="xs:string"/>
         <choice>
            <element name="Value" type="xs:string"/>
            <element name="EncodedValue" type="epm:QualifiedDataType"/>
         </choice>
      </sequence>
   </complexType>
   <complexType name="ClientApplicationType">
   <sequence>
      <element name="NameAndVersion" type="xs:string"/>
      <element name="ContentTransformScheme" type="epm:ContentTransformSchemeType" nillable="true"/>
   </sequence>
</complexType>
<complexType name="ContentTransformSchemeType">
   <simpleContent>
      <extension base="xs:string">
         <attribute name="ContentTransformSchemeURI" type="xs:anyURI" use="optional"/>
      </extension>
   </simpleContent>
</complexType>
<complexType name="IssuePostMarkedReceiptType">
   <sequence>
      <element name="Location" type="epm:ValidLocation" minoccurs="0"/>
      <element name="PostMarkImage" type="epm:PostMarkImageType"  minoccurs="0"/>
   </sequence>
</complexType>
<complexType name="PostMarkImageType">
   <simpleContent>
      <extension base="xs:boolean">
         <attribute name="Format" type="xs:string" default="JPG"/>
         <attribute name="Size" type="epm:ValidImageSize" default="Small"/>
      </extension>
   </simpleContent>
</complexType>
<complexType name="SignatureSelectorType">
   <sequence>
      <choice>
         <element name="NodeName" type="xs:string" maxOccurs="unbounded"/>
         <element name="XPathSelector" type="epm:XPathSelectorType"/>
      </choice>
   </sequence>
</complexType>
<complexType name="XPathSelectorType">
   <sequence>
      <element name="XPath" type="xs:string"/>
      <element name="NameSpace" type="xs:string" nillable="true"/>
      <element name="Qualifer" type="xs:string" nillable="true"/>
   </sequence>
</complexType>
<element name="VerifyRequest">
   <complexType>
      <sequence>
         <element name="VerifyOptions" type="epm:VerifyOptionsType"/>
         <element name="Version" type="xs:string"/>
         <element name="TransactionKey" type="epm:TransactionKeyType" nillable="true"/>
         <element name="ClaimedIdentity" type="epm:ClaimedIdentityType" nillable="true"/>
```

```
            <element name="OrganizationID" type="xs:string" nillable="true"/>
            <element name="ClientApplication" type="epm:ClientApplicationType"/>
            <element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <element name="SignatureData" type="epm:QualifiedDataType" nillable="true"/>
            <element name="SignedContent" type="epm:QualifiedDataType" nillable="true"/>
            <element ref="epm:PostMarkedReceipt" minOccurs="0"/>
            <element name="SignatureSelector" type="epm:SignatureSelectorType" nillable="true"/>
            <element name="ContentMetadata" type="epm:ContentMetadataType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="VerifyOptionsType">
      <sequence>
         <element name="EndLifecycle" type="xs:boolean"/>
         <element name="ExtendLifecycle" type="xs:boolean"/>
         <element name="VerifyCertificate" type="xs:boolean"/>
         <element name="DecryptIncomingEnvelope" type="xs:boolean"/>
         <element name="EncryptResponse" type="xs:boolean"/>
         <element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
         <element name="IssuePostMarkedReceipt" type="epm:IssuePostMarkedReceiptType" nillable="true"/>
         <element name="ReturnSignatureInfo" type="xs:boolean"/>
         <element name="ReturnX509Info" type="xs:boolean"/>
      </sequence>
   </complexType>
   <element name="VerifyResponse">
      <complexType>
         <sequence>
            <element name="TransactionStatus" type="xs:string"/>
            <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
            <element ref="epm:PostMarkedReceipt" minOccurs="0"/>
            <element name="SignatureData" type="epm:QualifiedDataType" nillable="true"/>
            <element name="SignatureInfo" type="epm:SignatureInfoType" nillable="true"/>
            <element name="X509Info" type="epm:X509InfoType" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <element name="PostMarkRequest">
      <complexType>
         <sequence>
            <element name="PostMarkOptions" type="epm:PostMarkOptionsType"/>
            <element name="Version" type="xs:string"/>
            <element name="SignatureType" type="epm:ValidSignatureType"/>
            <element name="TransactionKey" type="epm:TransactionKeyType" nillable="true"/>
            <element name="ClaimedIdentity" type="epm:ClaimedIdentityType" nillable="true"/>
            <element name="OrganizationID" type="xs:string" nillable="true"/>
            <element name="ClientApplication" type="epm:ClientApplicationType"/>
            <element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <element name="Data" type="epm:QualifiedDataType"/>
            <element name="ContentMetadata" type="epm:ContentMetadataType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="PostMarkOptionsType">
      <sequence>
         <element name="EndLifecycle" type="xs:boolean"/>
         <element name="ExtendLifecycle" type="xs:boolean"/>
         <element name="DecryptIncomingEnvelope" type="xs:boolean"/>
         <element name="EncryptResponse" type="xs:boolean"/>
         <element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
      </sequence>
   </complexType>
   <element name="PostMarkResponse">
      <complexType>
         <sequence>
            <element name="TransactionStatus" type="xs:string"/>
            <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
            <element ref="epm:PostMarkedReceipt"/>
            <element name="SignatureData" type="epm:QualifiedDataType" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <element name="CheckIntegrityRequest">
      <complexType>
         <sequence>
            <element name="CheckIntegrityOptions" type="epm:CheckIntegrityOptionsType"/>
            <element name="Version" type="xs:string"/>
            <element name="SignatureType" type="epm:ValidSignatureType" nillable="true"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
            <element name="ClaimedIdentity" type="epm:ClaimedIdentityType" nillable="true"/>
            <element name="OrganizationID" type="xs:string" nillable="true"/>
            <element name="ClientApplication" type="epm:ClientApplicationType"/>
            <element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <element name="OriginalContent" type="epm:OriginalContentType" maxOccurs="unbounded"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="CheckIntegrityOptionsType">
      <sequence>
         <element name="DecryptIncomingEnvelope" type="xs:boolean"/>
         <element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
         <element name="IssuePostMarkedReceipt" type="epm:IssuePostMarkedReceiptType" nillable="true"/>
      </sequence>
   </complexType>
   <element name="CheckIntegrityResponse">
      <complexType>
         <sequence>
            <element name="TransactionStatus" type="xs:string"/>
            <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
            <element ref="epm:PostMarkedReceipt" minOccurs="0"/>
         </sequence>
      </complexType>
   </element>
   <element name="LogEventRequest">
      <complexType>
         <sequence>
            <element name="LogEventOptions" type="epm:LogEventOptionsType"/>
            <element name="Version" type="xs:string"/>
            <element name="TransactionKey" type="epm:TransactionKeyType" nillable="true"/>
            <element name="ClaimedIdentity" type="epm:ClaimedIdentityType" nillable="true"/>
            <element name="OrganizationID" type="xs:string" nillable="true"/>
            <element name="ClientApplication" type="epm:ClientApplicationType"/>
            <element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <element name="Data" type="epm:QualifiedDataType"/>
            <element name="ContentMetadata" type="epm:ContentMetadataType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="LogEventOptionsType">
      <sequence>
         <element name="EndLifecycle" type="xs:boolean"/>
         <element name="ExtendLifecycle" type="xs:boolean"/>
         <element name="DecryptIncomingEnvelope" type="xs:boolean"/>
      </sequence>
   </complexType>
   <element name="LogEventResponse">
      <complexType>
         <sequence>
            <element name="TransactionStatus" type="xs:string"/>
            <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
         </sequence>
      </complexType>
   </element>
   <element name="StartLifecycleRequest">
      <complexType>
         <sequence>
```

```
            <element name="Version" type="xs:string"/>
            <element name="ClaimedIdentity" type="epm:ClaimedIdentityType" nillable="true"/>
            <element name="OrganizationID" type="xs:string" nillable="true"/>
            <element name="ClientApplication" type="epm:ClientApplicationType"/>
            <element name="ParticipatingParty" type="epm:ParticipatingPartyType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
            <element name="AccessScope" type="epm:Scopes"/>
         </sequence>
      </complexType>
   </element>
   <element name="StartLifecycleResponse">
      <complexType>
         <sequence>
            <element name="TransactionStatus" type="xs:string"/>
            <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
         </sequence>
      </complexType>
   </element>
   <element name="RetrieveResultsRequest">
      <complexType>
         <sequence>
            <element name="RetrieveResultsOptions" type="epm:RetrieveResultsOptionsType"/>
            <element name="Version" type="xs:string"/>
            <element name="SignatureType" type="epm:ValidSignatureType"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
            <element name="ClaimedIdentity" type="epm:ClaimedIdentityType" nillable="true"/>
            <element name="OrganizationID" type="xs:string" nillable="true"/>
            <element name="ClientApplication" type="epm:ClientApplicationType"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="RetrieveResultsOptionsType">
      <sequence>
         <element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
         <element name="IssuePostMarkedReceipt" type="epm:IssuePostMarkedReceiptType" nillable="true"/>
         <element name="EncryptResponse" type="xs:boolean"/>
         <element name="ReturnTimeStampAudit" type="xs:boolean"/>
         <element name="ReturnSignatureInfo" type="xs:boolean"/>
         <element name="ReturnX509Info" type="xs:boolean"/>
      </sequence>
   </complexType>
   <element name="RetrieveResultsResponse">
      <complexType>
         <sequence>
            <element name="TransactionStatus" type="xs:string"/>
            <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
```

```
minOccurs="0" maxOccurs="unbounded"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
            <element ref="epm:PostMarkedReceipt" minOccurs="0"/>
            <element name="Results" type="epm:ResultsType"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="ResultsType">
      <sequence>
         <element name="TransactionStatus" type="xs:string"/>
         <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
         <element name="TransactionKey" type="epm:TransactionKeyType"/>
         <element name="Operation" type="xs:string"/>
         <element name="OperationOptions" type="epm:ValidOption" nillable="true" minOccurs="0" maxOccurs="unbounded"/>
         <element name="OrganizationID" type="xs:string" nillable="true"/>
         <element name="UniqueSequenceId" type="xs:string" nillable="true"/>
         <element name="ClientApplication" type="epm:ClientApplicationType" nillable="true"/>
         <element name="ContentIdentifier" type="xs:string" nillable="true"/>
         <element ref="epm:PostMarkedReceipt" minOccurs="0"/>
         <element name="Data" type="epm:QualifiedDataType" nillable="true" minOccurs="0" maxOccurs="unbounded"/>
         <element name="ContentMetadata" type="epm:ContentMetadataType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
         <element name="TimeStampAudit" type="epm:QualifiedDataType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
         <element name="SignatureInfo" type="epm:SignatureInfoType" nillable="true"/>
         <element name="X509Info" type="epm:X509InfoType" nillable="true"/>
      </sequence>
   </complexType>
   <element name="SignRequest">
      <complexType>
         <sequence>
            <element name="SignOptions" type="epm:SignOptionsType"/>
            <element name="Version" type="xs:string"/>
            <element name="TransactionKey" type="epm:TransactionKeyType" nillable="true"/>
            <element name="ClaimedIdentity" type="epm:ClaimedIdentityType" nillable="true"/>
            <element name="OrganizationID" type="xs:string" nillable="true"/>
            <element name="ClientApplication" type="epm:ClientApplicationType"/>
            <element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <element name="Data" type="epm:QualifiedDataType"/>
            <element name="SignatureType" type="epm:ValidSignatureType"/>
            <element name="KeyName" type="xs:string" nillable="true"/>
            <element name="SignaturePolicyID" type="xs:anyURI" nillable="true"/>
            <element name="ContentMetadata" type="epm:ContentMetadataType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="SignOptionsType">
```

```
      <sequence>
         <element name="EndLifecycle" type="xs:boolean"/>
         <element name="ExtendLifecycle" type="xs:boolean"/>
         <element name="VerifyCertificate" type="xs:boolean"/>
         <element name="IssuePostMarkedReceipt" type="epm:IssuePostMarkedReceiptType" nillable="true"/>
         <element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
         <element name="DecryptIncomingEnvelope" type="xs:boolean"/>
         <element name="EncryptResponse" type="xs:boolean"/>
         <element name="ReturnSignatureInfo" type="xs:boolean"/>
         <element name="ReturnX509Info" type="xs:boolean"/>
      </sequence>
   </complexType>
   <element name="SignResponse">
      <complexType>
         <sequence>
            <element name="TransactionStatus" type="xs:string"/>
            <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
            <element ref="epm:PostMarkedReceipt" minOccurs="0"/>
            <element name="SignatureData" type="epm:QualifiedDataType" nillable="true"/>
            <element name="SignatureInfo" type="epm:SignatureInfoType" nillable="true"/>
            <element name="X509Info" type="epm:X509InfoType" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <element name="EncryptRequest">
      <complexType>
         <sequence>
            <element name="EncryptOptions" type="epm:EncryptOptionsType"/>
            <element name="Version" type="xs:string"/>
            <element name="TransactionKey" type="epm:TransactionKeyType" nillable="true"/>
            <element name="ClaimedIdentity" type="epm:ClaimedIdentityType" nillable="true"/>
            <element name="OrganizationID" type="xs:string" nillable="true"/>
            <element name="ClientApplication" type="epm:ClientApplicationType"/>
            <element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <element name="Data" type="epm:QualifiedDataType"/>
            <element name="SignatureType" type="epm:ValidSignatureType"/>
            <element name="NodeName" type="xs:string" nillable="true"/>
            <element name="SessionKeyAlgo" type="xs:string" nillable="true"/>
            <element name="CertificateSearchType" type="epm:ValidCertificateSearchType"/>
            <element name="CertificateID" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
            <element name="ContentMetadata" type="epm:ContentMetadataType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="EncryptOptionsType">
```

```
      <sequence>
         <element name="EndLifecycle" type="xs:boolean"/>
         <element name="ExtendLifecycle" type="xs:boolean"/>
         <element name="VerifyCertificate" type="xs:boolean"/>
         <element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
         <element name="IssuePostMarkedReceipt" type="epm:IssuePostMarkedReceiptType" nillable="true"/>
         <element name="DecryptIncomingEnvelope" type="xs:boolean"/>
         <element name="ReturnSignatureInfo" type="xs:boolean"/>
         <element name="ReturnX509Info" type="xs:boolean"/>
      </sequence>
   </complexType>
   <element name="EncryptResponse">
      <complexType>
         <sequence>
            <element name="TransactionStatus" type="xs:string"/>
            <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
            <element ref="epm:PostMarkedReceipt" minOccurs="0"/>
            <element name="SignatureData" type="epm:QualifiedDataType"/>
            <element name="SignatureInfo" type="epm:SignatureInfoType" nillable="true"/>
            <element name="X509Info" type="epm:X509InfoType" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <element name="DecryptRequest">
      <complexType>
         <sequence>
            <element name="DecryptOptions" type="epm:DecryptOptionsType"/>
            <element name="Version" type="xs:string"/>
            <element name="TransactionKey" type="epm:TransactionKeyType" nillable="true"/>
            <element name="ClaimedIdentity" type="epm:ClaimedIdentityType" nillable="true"/>
            <element name="OrganizationID" type="xs:string" nillable="true"/>
            <element name="ClientApplication" type="epm:ClientApplicationType"/>
            <element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <element name="EnvelopedData" type="epm:QualifiedDataType"/>
            <element name="ContentMetadata" type="epm:ContentMetadataType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="DecryptOptionsType">
      <sequence>
         <element name="EndLifecycle" type="xs:boolean"/>
         <element name="ExtendLifecycle" type="xs:boolean"/>
         <element name="StoreNonRepudiationEvidence" type="xs:boolean"/>
         <element name="EncryptResponse" type="xs:boolean"/>
         <element name="ReturnSignatureInfo" type="xs:boolean"/>
         <element name="ReturnX509Info" type="xs:boolean"/>
```

```
        </sequence>
    </complexType>
    <element name="DecryptResponse">
        <complexType>
            <sequence>
                <element name="TransactionStatus" type="xs:string"/>
                <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
                <element name="TransactionKey" type="epm:TransactionKeyType"/>
                <element name="Data" type="epm:QualifiedDataType"/>
                <element name="SignatureInfo" type="epm:SignatureInfoType" nillable="true"/>
                <element name="X509Info" type="epm:X509InfoType" nillable="true"/>
            </sequence>
        </complexType>
    </element>
    <element name="LocateRequest">
        <complexType>
            <sequence>
                <element name="LocateOptions" type="epm:LocateOptionsType"/>
                <element name="Version" type="xs:string"/>
                <element name="TransactionKey" type="epm:TransactionKeyType" nillable="true"/>
                <element name="ClaimedIdentity" type="epm:ClaimedIdentityType" nillable="true"/>
                <element name="OrganizationID" type="xs:string" nillable="true"/>
                <element name="ClientApplication" type="epm:ClientApplicationType"/>
                <element name="CertificateSearchType" type="epm:ValidCertificateSearchType"/>
                <element name="CertificateID" type="xs:string"/>
            </sequence>
        </complexType>
    </element>
    <complexType name="LocateOptionsType">
        <sequence>
            <element name="EndLifecycle" type="xs:boolean"/>
            <element name="ExtendLifecycle" type="xs:boolean"/>
            <element name="VerifyCertificate" type="xs:boolean"/>
            <element name="ReturnX509Info" type="xs:boolean"/>
        </sequence>
    </complexType>
    <element name="LocateResponse">
        <complexType>
            <sequence>
                <element name="TransactionStatus" type="xs:string"/>
                <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
                <element name="TransactionKey" type="epm:TransactionKeyType"/>
                <element name="PublicEncryptionCert" type="xs:string"/>
                <element name="X509Info" type="epm:X509InfoType" nillable="true"/>
            </sequence>
        </complexType>
```

```
   </element>
   <element name="RetrieveSummaryRequest">
      <complexType>
         <sequence>
            <element name="RetrieveSummaryOptions" type="epm:RetrieveSummaryOptionsType"/>
            <element name="Version" type="xs:string"/>
            <element name="TransactionKey" type="epm:TransactionKeyType" nillable="true"/>
            <element name="ClaimedIdentity" type="epm:ClaimedIdentityType" nillable="true"/>
            <element name="OrganizationID" type="xs:string" nillable="true"/>
            <element name="ClientApplication" type="epm:ClientApplicationType"/>
            <element name="ContentIdentifier" type="xs:string" nillable="true"/>
            <element name="LastUniqueSequenceId" type="xs:string" nillable="true"/>
            <element name="HashValue" type="xs:string" nillable="true"/>
            <element name="StartDateTime" type="xs:string" nillable="true"/>
            <element name="EndDateTime" type="xs:string" nillable="true"/>
            <element name="ContentMetadata" type="epm:ContentMetadataType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
            <element name="RetrieveCount" type="xs:string" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="RetrieveSummaryOptionsType">
      <sequence>
         <element name="EndLifecycle" type="xs:boolean"/>
         <element name="ExtendLifecycle" type="xs:boolean"/>
      </sequence>
   </complexType>
   <complexType name="RetrieveSummaryInfoType">
      <sequence>
         <element name="TransactionKey" type="epm:TransactionKeyType"/>
         <element name="UniqueSequenceId" type="xs:string" nillable="true"/>
         <element name="Operation" type="xs:string"/>
         <element name="OrganizationID" type="xs:string" nillable="true"/>
         <element name="ClientApplication" type="epm:ClientApplicationType" nillable="true"/>
         <element name="ContentIdentifier" type="xs:string" nillable="true"/>
         <element name="TransactionStatus" type="xs:string"/>
         <element name="TimeStampValue" type="xs:string" nillable="true"/>
         <element name="ContentHash" type="xs:string" nillable="true"/>
         <element name="SigningTime" type="xs:string" nillable="true"/>
         <element name="X509Subject" type="xs:string" nillable="true"/>
         <element name="X509Issuer" type="xs:string" nillable="true"/>
         <element name="X509Serial" type="xs:string" nillable="true"/>
      </sequence>
   </complexType>
   <element name="RetrieveSummaryResponse">
      <complexType>
         <sequence>
            <element name="TransactionStatus" type="xs:string"/>
            <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
```

```
minOccurs="0" maxOccurs="unbounded"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
            <element name="RetrieveSummaryInfo" type="epm:RetrieveSummaryInfoType" minOccurs="0"
maxOccurs="unbounded"/>
         </sequence>
      </complexType>
   </element>
   <element name="RetrievePostalAttributesRequest">
      <complexType>
         <sequence>
            <element name="Locator" type="epm:LocatorType"/>
            <element name="LanguageCode" type="xs:string"/>
            <element name="ClientApplication" type="epm:ClientApplicationType" nillable="true"/>
            <element name="AttributeCategory" type="xs:string" nillable="true"/>
         </sequence>
      </complexType>
   </element>
   <complexType name="PostalAttributeType">
      <sequence>
         <element name="AttributeName" type="xs:string"/>
         <element name="LastModifiedDateTime" type="xs:string"/>
         <element name="AttributeTextValue" type="xs:string" nillable="true"/>
         <element name="AttributeBinaryValue" type="xs:base64Binary" nillable="true"/>
      </sequence>
   </complexType>
   <element name="RetrievePostalAttributesResponse">
      <complexType>
         <sequence>
            <element name="TransactionStatus" type="xs:string"/>
            <element name="TransactionStatusDetail" type="epm:TransactionStatusDetailType" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
            <element name="TransactionKey" type="epm:TransactionKeyType"/>
            <element name="PostalAttribute" type="epm:PostalAttributeType" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
         </sequence>
      </complexType>
   </element>
</schema>
```

ANNEX D (INFORMATIVE) WEB SERVICES DESCRIPTION LANGUAGE (WSDL) FILE

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions name="EPMService"
       targetNamespace="http://www.upu.int/EPMService/definitions"
       xmlns="http://schemas.xmlsoap.org/wsdl/"
       xmlns:tns="http://www.upu.int/EPMService/definitions"
       xmlns:epm="http://www.upu.int/EPMService/schemas"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

<wsdl:types>

<xs:schema
   targetNamespace="http://www.upu.int/EPMService/schemas"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:epm="http://http://www.upu.int/EPMService/schemas">
   <xs:include schemaLocation="file://C:/EPM-UPU-WSDLs/EPMService-UPU-V1.15.xsd"/>
<!-- schemaLocation may be changed to reflect local needs as required.
Schema file may be placed on a Web Server and the above include can be changed to
http:// syntax as required. Also note that Visual Studio expects a conventional
directory path and file name in schemaLocation.
-->
</xs:schema>

</wsdl:types>


<!-- Request Response Message Definition -->

   <wsdl:message name="StartLifecycleRequestMessage">
      <wsdl:part name="StartLifecycleReq" element="epm:StartLifecycleRequest"/>
   </wsdl:message>

   <wsdl:message name="StartLifecycleResponseMessage">
      <wsdl:part name="StartLifecycleResp" element="epm:StartLifecycleResponse"/>
   </wsdl:message>

<!-- Request Response Message Definition -->

   <wsdl:message name="VerifyRequestMessage">
      <wsdl:part name="VerifyReq" element="epm:VerifyRequest"/>
   </wsdl:message>
   <wsdl:message name="VerifyResponseMessage">
      <wsdl:part name="VerifyResp" element="epm:VerifyResponse"/>
   </wsdl:message>

<!-- Request Response Message Definition -->

   <wsdl:message name="EncryptRequestMessage">
      <wsdl:part name="EncryptReq" element="epm:EncryptRequest"/>
   </wsdl:message>

   <wsdl:message name="EncryptResponseMessage">
      <wsdl:part name="EncryptResp" element="epm:EncryptResponse"/>
   </wsdl:message>

<!-- Request Response Message Definition -->

   <wsdl:message name="DecryptRequestMessage">
      <wsdl:part name="DecryptReq" element="epm:DecryptRequest"/>
   </wsdl:message>

   <wsdl:message name="DecryptResponseMessage">
      <wsdl:part name="DecryptResp" element="epm:DecryptResponse"/>
   </wsdl:message>

<!-- Request Response Message Definition -->

   <wsdl:message name="CheckIntegrityRequestMessage">
      <wsdl:part name="CheckIntegrityReq" element="epm:CheckIntegrityRequest"/>
   </wsdl:message>

   <wsdl:message name="CheckIntegrityResponseMessage">
      <wsdl:part name="CheckIntegrityResp" element="epm:CheckIntegrityResponse"/>
   </wsdl:message>

<!-- Request Response Message Definition -->

   <wsdl:message name="RetrieveResultsRequestMessage">
      <wsdl:part name="RetrieveResultsReq" element="epm:RetrieveResultsRequest"/>
   </wsdl:message>

   <wsdl:message name="RetrieveResultsResponseMessage">
      <wsdl:part name="RetrieveResultsResp" element="epm:RetrieveResultsResponse"/>
   </wsdl:message>

<!-- Request Response Message Definition -->

   <wsdl:message name="RetrieveSummaryRequestMessage">
```

```
      <wsdl:part name="RetrieveSummaryReq" element="epm:RetrieveSummaryRequest"/>
   </wsdl:message>

   <wsdl:message name="RetrieveSummaryResponseMessage">
      <wsdl:part name="RetrieveSummaryResp" element="epm:RetrieveSummaryResponse"/>
   </wsdl:message>

<!-- Request Response Message Definition -->

   <wsdl:message name="LocateRequestMessage">
      <wsdl:part name="LocateReq" element="epm:LocateRequest"/>
   </wsdl:message>

   <wsdl:message name="LocateResponseMessage">
      <wsdl:part name="LocateResp" element="epm:LocateResponse"/>
   </wsdl:message>

<!-- Request Response Message Definition -->

   <wsdl:message name="PostMarkRequestMessage">
      <wsdl:part name="PostMarkReq" element="epm:PostMarkRequest"/>
   </wsdl:message>

   <wsdl:message name="PostMarkResponseMessage">
      <wsdl:part name="PostMarkResp" element="epm:PostMarkResponse"/>
   </wsdl:message>

<!-- Request Response Message Definition -->

   <wsdl:message name="SignRequestMessage">
      <wsdl:part name="SignReq" element="epm:SignRequest"/>
   </wsdl:message>

   <wsdl:message name="SignResponseMessage">
      <wsdl:part name="SignResp" element="epm:SignResponse"/>
   </wsdl:message>

<!-- Request Response Message Definition -->

   <wsdl:message name="LogEventRequestMessage">
      <wsdl:part name="LogEventReq" element="epm:LogEventRequest"/>
   </wsdl:message>

   <wsdl:message name="LogEventResponseMessage">
      <wsdl:part name="LogEventResp" element="epm:LogEventResponse"/>
   </wsdl:message>

<!-- Request Response Message Definition -->
```

```
   <wsdl:message name="RetrievePostalAttributesRequestMessage">
      <wsdl:part name="RetrievePostalAttributesReq" element="epm:RetrievePostalAttributesRequest"/>
   </wsdl:message>

   <wsdl:message name="RetrievePostalAttributesResponseMessage">
      <wsdl:part name="RetrievePostalAttributesResp" element="epm:RetrievePostalAttributesResponse"/>
   </wsdl:message>


<!-- WSDL schema PortType and Operation to Message mapping elements -->

<wsdl:portType name="EPMServicePortType">

   <wsdl:operation name="StartLifecycle">
      <input message="tns:StartLifecycleRequestMessage"/>
      <output message="tns:StartLifecycleResponseMessage"/>
   </wsdl:operation>

   <wsdl:operation name="Verify">
      <input message="tns:VerifyRequestMessage"/>
      <output message="tns:VerifyResponseMessage"/>
   </wsdl:operation>

   <wsdl:operation name="Encrypt">
      <input message="tns:EncryptRequestMessage"/>
      <output message="tns:EncryptResponseMessage"/>
   </wsdl:operation>

   <wsdl:operation name="Decrypt">
      <input message="tns:DecryptRequestMessage"/>
      <output message="tns:DecryptResponseMessage"/>
   </wsdl:operation>

   <wsdl:operation name="CheckIntegrity">
      <input message="tns:CheckIntegrityRequestMessage"/>
      <output message="tns:CheckIntegrityResponseMessage"/>
   </wsdl:operation>

   <wsdl:operation name="RetrieveResults">
      <input message="tns:RetrieveResultsRequestMessage"/>
      <output message="tns:RetrieveResultsResponseMessage"/>
   </wsdl:operation>

   <wsdl:operation name="RetrieveSummary">
      <input message="tns:RetrieveSummaryRequestMessage"/>
      <output message="tns:RetrieveSummaryResponseMessage"/>
   </wsdl:operation>
   <wsdl:operation name="Locate">
      <input message="tns:LocateRequestMessage"/>
      <output message="tns:LocateResponseMessage"/>
   </wsdl:operation>

   <wsdl:operation name="PostMark">
      <input message="tns:PostMarkRequestMessage"/>
      <output message="tns:PostMarkResponseMessage"/>
   </wsdl:operation>

   <wsdl:operation name="Sign">
      <input message="tns:SignRequestMessage"/>
      <output message="tns:SignResponseMessage"/>
   </wsdl:operation>

   <wsdl:operation name="LogEvent">
      <input message="tns:LogEventRequestMessage"/>
      <output message="tns:LogEventResponseMessage"/>
   </wsdl:operation>

   <wsdl:operation name="RetrievePostalAttributes">
      <input message="tns:RetrievePostalAttributesRequestMessage"/>
      <output message="tns:RetrievePostalAttributesResponseMessage"/>
   </wsdl:operation>

</wsdl:portType>

<!-- WSDL schema Binding and Operations elements -->

<wsdl:binding name="EPMServiceBinding" type="tns:EPMServicePortType">
   <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>

   <wsdl:operation name="StartLifecycle">
      <soap:operation soapAction="StartLifecycle"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="Verify">
      <soap:operation soapAction="Verify"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
</wsdl:operation>

<wsdl:operation name="Encrypt">
   <soap:operation soapAction="Encrypt"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
</wsdl:operation>

<wsdl:operation name="Decrypt">
   <soap:operation soapAction="Decrypt"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
</wsdl:operation>

<wsdl:operation name="CheckIntegrity">
   <soap:operation soapAction="CheckIntegrity"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
</wsdl:operation>

<wsdl:operation name="RetrieveResults">
   <soap:operation soapAction="RetrieveResults"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
</wsdl:operation>

<wsdl:operation name="RetrieveSummary">
   <soap:operation soapAction="RetrieveSummary"/>
      <wsdl:input>
         <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="Locate">
      <soap:operation soapAction="Locate"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="PostMark">
      <soap:operation soapAction="PostMark"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="Sign">
      <soap:operation soapAction="Sign"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="LogEvent">
      <soap:operation soapAction="LogEvent"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="RetrievePostalAttributes">
      <soap:operation soapAction="RetrievePostalAttributes"/>
         <wsdl:input>
```

```
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
   </wsdl:operation>


</wsdl:binding>

<!-- WSDL schema Service and Location elements -->

<wsdl:service name="EPMService">
   <wsdl:port name="EPMServicePort" binding="tns:EPMServiceBinding">
      <soap:address location="http://www.myPostalAdmin.com:8000/EPMService"/>
   </wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

Bibliography

This bibliography provides full reference and sourcing information for all standards and other reference sources which are quoted in the above text. For references which mention specific version numbers or dates, subsequent amendments to, or revisions of, any of these publications might not be relevant. However, users of this document are encouraged to investigate the existence and applicability of more recent editions. For references without date or version number, the latest edition of the document referred to applies. It is stressed that only referenced documents are listed here.

Internet Engineering Task Force Public Key Infrastructure X.509 working group (IETF PKIX) documents.

NOTE 1 Available at www.ietf.org

IETF Lightweight Directory Access Protocol (LDAP). This standard is adopted for data dissemination.

Certificate Profile is the IETF certificate profile as described by the IETF PKIX working group, ref RFC 2459 – Internet X.509 Public Key Infrastructure Certificate and CRL Profile (January 1999)

Certificate Revocation List profile is the IETF CRL profile as described by the IETF PKIX working group, ref. RFC 2459 – Internet X.509 Public Key Infrastructure Certificate and CRL Profile (January 1999)

[1] RFC 2459 – Internet X.509 Public Key Infrastructure: Certificate and CRL Profile (January 1999), R. Housley, W. Ford, W. Polk, D. Solo

[2] RFC 2510 – Internet X.509 Public Key Infrastructure: Certificate Management Protocols (March 1999), C. Adams, S. Farrel

[3] RFC 2511 – Internet X.509: Certificate Request Message Format (March 1999), M. Myers, C. Adams, D. Solo, D. Kemp

[4] RFC 2527 – Internet X.509 Public Key Infrastructure: Certificate Policy and Certification Practices Framework (March 1999), S. Chockani, W. Ford

[5] RFC 2528 – Internet X.509 Public Key Infrastructure: Representation of Key Exchange Algorithm (KEA) Keys in Internet X.509 Public Key Infrastructure Certificates (March 1999), R. Housley, W. Polk

[6] RFC 2630 – Cryptographic Message Syntax (June 1999), R. Housley

NOTE 2 Defines the ASN.1 layout for all relevant PKCS objects utilised by the EPM.

[7] RFC 3126 – Electronic Signature Formats for long term electronic signatures (September 2001), D. Pinkas, J. Ross, N. Pope

NOTE 3 This RFC is considered by most authorities to be the essential definition of what constitutes a legitimate non-repudiation service. It describes the technical criteria and pre-requisites for minimum compliance as a non-repudiation service capability. The EPM interface specification honours all mandatory requirements, i.e. qualified as "required" or "shall" in the text.

[8] RFC 3275 – XML-Signature Syntax and Processing (March 2002), D. Eastlake, J. Reagle, D. Solo

NOTE 4 This RFC, commonly referred to as XMLDSIG, is the W3C's landmark standard to which nearly all XML-based attempts to capture ASN.1 PKCS7 syntax in XML refer. The current WSDL interface allows for both PKCS7 binary ASN.1 formatting of signature objects as well as the more recent XMLDSIG signature formatting.

European Telecommunications Standards Institute (ETSI) Standards

NOTE 5 ETSI standards are available at www.etsi.org.

[9] ETSI TS 101 733 – Electronic Signatures and Infrastructures (ESI) – Electronic Signature Formats (8 December 2000)

NOTE 6 This is the European equivalent of RFC 3126 and is also honoured by the EPM interface specification.

[10] ETSI TS 101 903 – XML Advanced Electronic Signatures (XAdES) (12 February 2002)

NOTE 7 This is the XML equivalent of TS 101 733 above, and although not currently implemented in the EPM reference implementation, is a clearly desirable target which will eventually be implemented in addition to the current ASN.1 based implementation. It is the intention of the EPM Interface to support the predominate XML Digital Signature standards which prevail in the market place. At the time of this writing the only clear consensus on digital signature representation in XML is RFC 3275, i.e. XMLDSIG.